Phase 1: User Database and Account System & Phase 2: Personalized Property Recommendation System

Let's start by implementing the outlined changes for Phase 1 and 2.

Table of Contents

Outline

Phase 1: User Database and Account System

- Create Database Tables
- Implement the Database Context
- Configure Database Connection in `Startup.cs`
- Implement User Registration in `AccountController.cs`
- Implement UserService

Phase 2: Personalized Property Recommendation System

- Design Recommendation Engine Architecture
- Deployment and Monitoring
- Documentation and User Training

Phase 1: User Database and Account System

1. Create Database Tables

SQL Script to Create Tables:

```
-- Create Users table
CREATE TABLE Users (
```

```
    Id INT PRIMARY KEY IDENTITY,
    Username NVARCHAR(50) NOT NULL UNIQUE,
    Email NVARCHAR(100) NOT NULL UNIQUE,
    PasswordHash NVARCHAR(100) NOT NULL,
    CreatedAt DATETIME2 NOT NULL DEFAULT GETUTCDATE()
);

-- Create Favorites table
CREATE TABLE Favorites (
    Id INT PRIMARY KEY IDENTITY,
    UserId INT NOT NULL,
    PropertyId INT NOT NULL,
    FOREIGN KEY (UserId) REFERENCES Users(Id),
    FOREIGN KEY (PropertyId) REFERENCES Properties(Id)
);
```

2. Implement the Database Context

AppDbContext.cs:

```
using Microsoft.EntityFrameworkCore;
using NexHomeAgentAPIv2.Models;

namespace NexHomeAgentAPIv2
{
    public class AppDbContext : DbContext
    {
        public AppDbContext(DbContextOptions<AppDbContext> op
tions) : base(options) { }

        public DbSet<User> Users { get; set; }
        public DbSet<Favorite> Favorites { get; set; }
        public DbSet<NexHomeAgentAPIv2.Models.Property> Prope
rties { get; set; }
```

```
    }
}
```

3. Configure Database Connection in `Startup.cs`

Startup.cs:

```
public void ConfigureServices(IServiceCollection services)
{
    // Add database context
    services.AddDbContext<AppDbContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionStrin
g("DefaultConnection")));

    // Add other services
    services.AddControllers();
    services.AddScoped<IUserService, UserService>();
    services.AddScoped<IPasswordHasher<User>, PasswordHasher<
User>>();

    // Add authentication and authorization
    services.AddAuthentication(JwtBearerDefaults.Authenticati
onScheme)
        .AddJwtBearer(options =>
        {
            options.TokenValidationParameters = new TokenVali
dationParameters
            {
                ValidateIssuer = true,
                ValidateAudience = true,
                ValidateLifetime = true,
                ValidateIssuerSigningKey = true,
                ValidIssuer = Configuration["Jwt:Issuer"],
                ValidAudience = Configuration["Jwt:Issuer"],
                IssuerSigningKey = new SymmetricSecurityKey(E
ncoding.UTF8.GetBytes(Configuration["Jwt:Key"]))
```

```
            };
        });

    services.AddAuthorization();
}
```

4. Implement User Registration in `AccountController.cs`

AccountController.cs:

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Identity;
using Microsoft.Extensions.Configuration;
using System.Threading.Tasks;
using System.Linq;
using System;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using Microsoft.IdentityModel.Tokens;
using System.Text;

[ApiController]
[Route("api/[controller]")]
public class AccountController : ControllerBase
{
    private readonly IUserService _userService;
    private readonly IPasswordHasher<User> _passwordHasher;
    private readonly IConfiguration _configuration;

    public AccountController(IUserService userService, IPassw
ordHasher<User> passwordHasher, IConfiguration configuration)
    {
        _userService = userService;
        _passwordHasher = passwordHasher;
        _configuration = configuration;
    }
```

```
    [HttpPost("register")]
    public async Task<IActionResult> Register(RegisterModel m
odel)
    {
        if (!ModelState.IsValid)
        {
            return BadRequest(ModelState);
        }

        if (await _userService.UserExistsAsync(model.Usernam
e, model.Email))
        {
            return BadRequest("Username or email is already t
aken.");
        }

        var user = new User
        {
            Username = model.Username,
            Email = model.Email,
            PasswordHash = _passwordHasher.HashPassword(null,
model.Password)
        };

        await _userService.CreateUserAsync(user);

        return Ok();
    }

    [HttpPost("login")]
    public async Task<IActionResult> Login(LoginModel model)
    {
        if (!ModelState.IsValid)
        {
            return BadRequest(ModelState);
```

```
        }

        var user = await _userService.GetUserByUsernameAsync
(model.Username);

        if (user == null || _passwordHasher.VerifyHashedPassw
ord(null, user.PasswordHash, model.Password) == PasswordVerif
icationResult.Failed)
        {
            return BadRequest("Invalid username or passwor
d.");
        }

        var token = GenerateJwtToken(user);
        return Ok(new { Token = token });
    }

    private string GenerateJwtToken(User user)
    {
        var securityKey = new SymmetricSecurityKey(Encoding.U
TF8.GetBytes(_configuration["Jwt:Key"]));
        var credentials = new SigningCredentials(securityKey,
SecurityAlgorithms.HmacSha256);

        var claims = new[]
        {
            new Claim(JwtRegisteredClaimNames.Sub, user.Usern
ame),
            new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGu
id().ToString())
        };

        var token = new JwtSecurityToken(
            issuer: _configuration["Jwt:Issuer"],
            audience: _configuration["Jwt:Issuer"],
            claims: claims,
```

```
            expires: DateTime.Now.AddMinutes(120),
            signingCredentials: credentials);

        return new JwtSecurityTokenHandler().WriteToken(toke
n);
    }

    [HttpPost("change-password")]
    public async Task<IActionResult> ChangePassword(ChangePas
swordModel model)
    {
        if (!ModelState.IsValid)
        {
            return BadRequest(ModelState);
        }

        var userId = User.FindFirst(JwtRegisteredClaimNames.S
ub)?.Value;
        var user = await _userService.GetUserByIdAsync(int.Pa
rse(userId));

        if (user == null || _passwordHasher.VerifyHashedPassw
ord(null, user.PasswordHash, model.CurrentPassword) == Passwo
rdVerificationResult.Failed)
        {
            return BadRequest("Invalid current password.");
        }

        user.PasswordHash = _passwordHasher.HashPassword(nul
l, model.NewPassword);
        await _userService.UpdateUserAsync(user);

        return Ok();
    }
}
```

5. Implement UserService

UserService.cs:

```
using Microsoft.EntityFrameworkCore;
using System.Threading.Tasks;

public interface IUserService
{
    Task<bool> UserExistsAsync(string username, string emai
l);
    Task CreateUserAsync(User user);
    Task<User> GetUserByUsernameAsync(string username);
    Task<User> GetUserByIdAsync(int userId);
    Task UpdateUserAsync(User user);
}

public class UserService : IUserService
{
    private readonly AppDbContext _context;

    public UserService(AppDbContext context)
    {
        _context = context;
    }

    public async Task<bool> UserExistsAsync(string username,
string email)
    {
        return await _context.Users.AnyAsync(u => u.Username
== username || u.Email == email);
    }

    public async Task CreateUserAsync(User user)
    {
        _context.Users.Add(user);
```

```
            await _context.SaveChangesAsync();
        }

        public async Task<User> GetUserByUsernameAsync(string use
rname)
        {
            return await _context.Users.SingleOrDefaultAsync(u =>
u.Username == username);
        }

        public async Task<User> GetUserByIdAsync(int userId)
        {
            return await _context.Users.FindAsync(userId);
        }

        public async Task UpdateUserAsync(User user)
        {
            _context.Users.Update(user);
            await _context.SaveChangesAsync();
        }
    }
```

Updated Notion List

Phase 1: User Database and Account System

- Create Database Tables
- Implement the Database Context
- Configure Database Connection in `Startup.cs`
- Implement User Registration in `AccountController.cs`
- Implement UserService
- Add Model Classes

Model Classes:

User.cs:

```
namespace NexHomeAgentAPIv2.Models
{
    public class User
    {
        public int Id { get; set; }
        public string Username { get; set; } = string.Empty;
        public string Email { get; set; } = string.Empty;
        public string PasswordHash { get; set; } = string.Emp
ty;
        public DateTime CreatedAt { get; set; }
    }
}
```

LoginModel.cs:

```
namespace NexHomeAgentAPIv2.Models
{
    public class LoginModel
    {
        public string Username { get; set; } = string.Empty;
        public string Password { get; set; } = string.Empty;
    }
}
```

RegisterModel.cs:

```
namespace NexHomeAgentAPIv2.Models
{
    public class RegisterModel
```

```
    {
        public string Username { get; set; } = string.Empty;
        public string Email { get; set; } = string.Empty;
        public string Password { get; set; } = string.Empty;
    }
}
```

Property.cs:

```
namespace NexHomeAgentAPIv2.Models
{
    public class Property
    {
        public int Id { get; set; }
        public string Address { get; set; } = string.Empty;
        public string City { get; set; } = string.Empty;
        public string State { get; set; } = string.Empty;
        public string ZipCode { get; set; } = string.Empty;
        public decimal Price { get; set; }
        public int Bedrooms { get; set; }
        public int Bathrooms { get; set; }
        public int SquareFeet { get; set; }
        public string Description { get; set; } = string.Empt
y;
    }
}
```

Favorite.cs:

```
namespace NexHomeAgentAPIv2.Models
{
    public class Favorite
```

```
    {
        public int Id { get; set; }
        public int UserId { get; set; }
        public int PropertyId { get; set; }
        public User User { get; set; } = new User();
        public Property Property { get; set; } = new Property
();
    }
}
```

Phase 2: Personalized Property Recommendation System

1. Design Recommendation Engine Architecture

Hybrid Recommendation System Implementation:

```
from flask import Flask, request, jsonify
import xgboost as xgb
import pandas as pd
import lightgbm as lgb
from sklearn.neighbors import NearestNeighbors
import random
from sklearn.metrics import precision_score, recall_score, nd
cg_score
import numpy as np

app = Flask(__name__)

# Load models and data
model = xgb.XGBRegressor()
model.load_model('new_property_valuation_model.json')
property_features = pd.read_csv('property_features.csv')
collaborative_recommender = NearestNeighbors(n_neighbors=5)
collaborative_recommender.fit(property_features)
content_recommender = lgb.LGBMClassifier()
```

```
content_recommender.fit(property_features.drop(columns=['prop
erty_id']), property_features['property_id'])

user_behavior = pd.read_csv('user_behavior.csv')

@app.route('/recommend', methods=['POST'])
def recommend():
    try:
        user_id = request.json['user_id']
        user_data = user_behavior[user_behavior['user_id'] ==
user_id]

        test_variant = random.choice(['hybrid', 'collaborativ
e', 'content-based'])

        if test_variant == 'hybrid':
            collab_recommendations = collaborative_recommende
r.kneighbors(user_data, return_distance=False)
            content_recommendations = content_recommender.pre
dict(user_data)
            combined_recommendations = list(set(collab_recomm
endations[0]) | set(content_recommendations))
            recommended_properties = property_features.iloc[c
ombined_recommendations].to_dict(orient='records')
        elif test_variant == 'collaborative':
            collab_recommendations = collaborative_recommende
r.kneighbors(user_data, return_distance=False)
            recommended_properties = property_features.iloc[c
ollab_recommendations[0]].to_dict(orient='records')
        else:
            content_recommendations = content_recommender.pre
dict(user_data)
            recommended_properties = property_features.iloc[c
ontent_recommendations].to_dict(orient='records')

        return jsonify({'recommendations': recommended_proper
```

```
ties, 'test_variant': test_variant})
    except Exception as e:
        return jsonify({'error': str(e)}), 500

@app.route('/feedback', methods=['POST'])
def provide_feedback():
    try:
        feedback = request.get_json()
        user_id = feedback['user_id']
        property_id = feedback['property_id']
        rating = feedback['rating']

        user_behavior.append({'user_id': user_id, 'property_i
d': property

_id, 'rating': rating}, ignore_index=True)
        user_behavior.to_csv('user_behavior.csv', index=Fals
e)

        collaborative_recommender.fit(property_features)
        content_recommender.fit(property_features.drop(column
s=['property_id']), property_features['property_id'])

        return jsonify({'status': 'Feedback received and mode
ls updated.'}), 200
    except Exception as e:
        return jsonify({'error': str(e)}), 500

def offline_evaluation():
    try:
        train_data, test_data = train_test_split(user_behavio
r, test_size=0.2, random_state=42)
        collaborative_recommender.fit(property_features)
        content_recommender.fit(property_features.drop(column
s=['property_id']), property_features['property_id'])
```

```
        precision = []
        recall = []
        ndcg = []

        for _, user_data in test_data.groupby('user_id'):
            collab_recommendations = collaborative_recommende
r.kneighbors(user_data, return_distance=False)
            content_recommendations = content_recommender.pre
dict(user_data)
            combined_recommendations = list(set(collab_recomm
endations[0]) | set(content_recommendations))

            true_properties = user_data['property_id'].tolist
()
            precision.append(precision_score(true_properties,
combined_recommendations, top_k=5))
            recall.append(recall_score(true_properties, combi
ned_recommendations, top_k=5))
            ndcg.append(ndcg_score([true_properties], [combin
ed_recommendations], k=5))

        print(f"Precision@5: {np.mean(precision)}")
        print(f"Recall@5: {np.mean(recall)}")
        print(f"NDCG@5: {np.mean(ndcg)}")
    except Exception as e:
        print(f"Error in offline evaluation: {e}")

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

2. Deployment and Monitoring

Deploy the Flask App as a Web Service:

1. Create a Dockerfile for the Flask app:

```
FROM python:3.9-slim

WORKDIR /app

COPY requirements.txt requirements.txt
RUN pip install -r requirements.txt

COPY . .

CMD ["python", "app.py"]
```

1. Build and push the Docker image:

```
docker build -t yourdockerhubusername/recommendation-service:
latest .
docker push yourdockerhubusername/recommendation-service:late
st
```

1. Deploy the Docker container to Azure:

```
az container create --resource-group yourResourceGroup --name
recommendation-service --image yourdockerhubusername/recommen
dation-service:latest --cpu 1 --memory 1 --ports 5000
```

Set Up Monitoring with Azure Application Insights:

1. Install the Azure Application Insights SDK for Python:

```
pip install opencensus-ext-azure
```

1. Integrate Application Insights into the Flask app:

```
from opencensus.ext.azure.log_exporter import AzureLogHandler
import logging
```

```
logging.basicConfig(level=logging.INFO)
logging.getLogger().addHandler(AzureLogHandler(connection_str
ing="InstrumentationKey=your-instrumentation-key-here"))
```

3. Documentation and User Training

Documentation for AI Integration:

```
# AI Integration Documentation

## Property Valuation Model
- **Model:** XGBoost
- **Data Sources:** Neighborhood demographics, school distric
t info, market trends
- **Features:** Price per sqft, age of property, etc.

## Recommendation System
- **Algorithm:** Nearest Neighbors and LightGBM
- **Data:** User behavior, property features
- **A/B Testing:** Implemented to compare hybrid, collaborati
ve, and content-based recommendation approaches
- **Offline Evaluation:** Assessed using precision, recall, a
nd NDCG metrics
```

User Training Guide:

```
# User Guide for New AI Features

## Property Valuation
- **Using the Valuation Tool:** Step-by-step instructions on
using the property valuation tool.
- **Understanding Valuation Results:** Explanation of the val
uation metrics and how to interpret them.

## Recommendation System
- **How it Works:** Description of the hybrid recommendation
```

```
system combining collaborative and content-based filtering.
- **Using Recommendations:** How to view and use property rec
ommendations.
- **Providing Feedback:** How to provide feedback to improve
recommendations.
- **A/B Testing:** Understanding the purpose of A/B testing a
nd how it impacts recommendations.
```

Summary

By implementing these changes, we will have a robust user account management system, a hybrid recommendation system, and comprehensive monitoring and documentation. This will enhance the NexHomeAgent application's functionality and user experience.

Would you like to proceed with the implementation, or is there any specific area you would like to focus on first?