Phase 2 Install dependentcys: User Database & Account System.

Outline

1. Introduction
2. Create `requirements.txt` for Python Dependencies
3. Install Python Dependencies
4. Update `Startup.cs` for .NET Dependencies
5. Install .NET Dependencies
6. Apply Migrations and Update Database
7. Final Setup
8. Summary

Table of Contents

- **Script to Install Dependencies for Phase 2 and User Database & Account System**
- **Table of Contents**
  - Step 1: Create `requirements.txt` for Python Dependencies
  - Step 2: Install Python Dependencies
  - Step 3: Update `Startup. cs` for .NET Dependencies
  - Step 4: Install .NET Dependencies
  - Step 5: Apply Migrations and Update Database
  - Step 6: Final Setup
  - Summary
- **Step 1: Create `requirements.txt` for Python Dependencies**
- **Step 2: Install Python Dependencies**
- **Step 3: Update `Startup. cs` for .NET Dependencies**
- **Step 4: Install .NET Dependencies**
- **Step 5: Apply Migrations and Update Database**
- **Step 6: Final Setup**

- **Summary.**

Table of Contents

Script to Install Dependencies for Phase 2 and User Database & Account System

To set up the necessary dependencies for Phase 2 and the user database/account system, we'll create a `requirements.txt` file for the Python dependencies and update the `Startup.cs` for the .NET dependencies.

Step 1: Create `requirements.txt` for Python Dependencies

Create a `requirements.txt` file with the following content:

```
Flask==2.0.3
xgboost==1.5.2
pandas==1.3.5
lightgbm==3.3.2
scikit-learn==1.0.2
opencensus-ext-azure==1.0.7
```

Step 2: Install Python Dependencies

Run the following commands to create a virtual environment and install the dependencies:

```
# Create a virtual environment
python -m venv venv

# Activate the virtual environment
# On Windows
venv\\Scripts\\activate
# On macOS/Linux
source venv/bin/activate

# Install dependencies
pip install -r requirements.txt
```

Step 3: Update `Startup.cs` for .NET Dependencies

Make sure your `Startup.cs` file includes the necessary services for Entity Framework Core, authentication, and dependency injection.

```
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.IdentityModel.Tokens;
using System.Text;

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

    public IConfiguration Configuration { get; }

    public void ConfigureServices(IServiceCollection service
s)
```

```
    {
        // Add database context
        services.AddDbContext<AppDbContext>(options =>
            options.UseSqlServer(Configuration.GetConnectionS
tring("DefaultConnection")));

        // Add authentication and authorization
        services.AddAuthentication(JwtBearerDefaults.Authenti
cationScheme)
            .AddJwtBearer(options =>
            {
                options.TokenValidationParameters = new Token
ValidationParameters
                {
                    ValidateIssuer = true,
                    ValidateAudience = true,
                    ValidateLifetime = true,
                    ValidateIssuerSigningKey = true,
                    ValidIssuer = Configuration["Jwt:Issue
r"],
                    ValidAudience = Configuration["Jwt:Issue
r"],
                    IssuerSigningKey = new SymmetricSecurityK
ey(Encoding.UTF8.GetBytes(Configuration["Jwt:Key"]))
                };
            });

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

        // Add controllers
        services.AddControllers();
    }
```

```
    public void Configure(IApplicationBuilder app, IWebHostEn
vironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
        }

        app.UseRouting();
        app.UseAuthentication();
        app.UseAuthorization();
        app.UseEndpoints(endpoints =>
        {
            endpoints.MapControllers();
        });
    }
}
```

Step 4: Install .NET Dependencies

Run the following commands in the terminal to install the necessary .NET dependencies:

```
dotnet add package Microsoft.EntityFrameworkCore.SqlServer
dotnet add package Microsoft.AspNetCore.Authentication.JwtBea
rer
dotnet add package Microsoft.Extensions.Configuration
dotnet add package Microsoft.Extensions.Configuration.Json
dotnet add package Microsoft.AspNetCore.Identity
dotnet add package System.IdentityModel.Tokens.Jwt
```

Step 5: Apply Migrations and Update Database

Run the following commands to apply migrations and update the database:

```
dotnet ef migrations add InitialCreate
```

```
dotnet ef database update
```

Step 6: Final Setup

Ensure your application settings (`appsettings.json`) include the necessary configuration for the database connection and JWT authentication:

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=<server-name>.database.windo
ws.net;Database=<database-name>;User Id=<username>;Password=<
password>;"
  },
  "Jwt": {
    "Key": "your_secret_key_here",
    "Issuer": "your_issuer_here"
  }
}
```

Summary

By following these steps, you will set up the environment and install the necessary dependencies for Phase 2 and the user database/account system in Visual Studio. This will prepare your project for development and deployment.

If you have any questions or need further assistance, please let me know!