Real-Time Title Document Verification & Risk Assessment Platform

Step-by-Step Backend Setup (U.S. Mortgage Focused)

Tech Stack:
- Frontend: React.js + Material UI
- Backend: ASP.NET Core Web API (C#)
- Database: SQL Server or PostgreSQL
- Authentication: JWT-based login
- File Storage: Local (dev), Azure Blob (prod)
- DevOps: Docker, GitHub Actions, Azure/AWS
- Testing: Swagger, Postman

Document Types:
- Title Deeds
- Sale Agreements
- Encumbrance Certificates
- Survey Maps

Risk Parameters:
- Ownership chain clarity
- Liens and encumbrances
- Tax delinquency
- Survey map conflicts
- Legal disputes
- Parcel ID validity
- Stamp duty compliance
- OCR confidence
- Document age

Risk Score: 1·10 (Green: 1·3, Yellow: 4·6, Red: 7·10)

Directory Structure:
/TitleVerificationPlatform.sln
⋯ /TitleVerification.API · ASP.NET Core Web API
⋯ /TitleVerification.Models · Shared DTOs and Entities
⋯ /TitleVerification.Services · OCR, Risk Logic

Required NuGet Packages:
- Microsoft.AspNetCore.Authentication.JwtBearer
- Microsoft.IdentityModel.Tokens
- System.IdentityModel.Tokens.Jwt
- Microsoft.EntityFrameworkCore.SqlServer
- Microsoft.EntityFrameworkCore.Tools
- Swashbuckle.AspNetCore

Install via CLI:
dotnet add package Microsoft.AspNetCore.Authentication.JwtBearer
dotnet add package Microsoft.IdentityModel.Tokens
dotnet add package System.IdentityModel.Tokens.Jwt
dotnet add package Microsoft.EntityFrameworkCore.SqlServer
dotnet add package Microsoft.EntityFrameworkCore.Tools
dotnet add package Swashbuckle.AspNetCore

Models (TitleVerification.Models):

Document.cs:

```
public class Document
{
    public int Id { get; set; }
    public string FileName { get; set; }
    public string DocumentType { get; set; }
    public string OwnerName { get; set; }
    public string PropertyAddress { get; set; }
    public string ParcelId { get; set; }
    public DateTime DateOfIssue { get; set; }
    public string UploadedBy { get; set; }
    public DateTime UploadedAt { get; set; }
    public int RiskScore { get; set; }
    public string RiskColor { get; set; }
    public string RiskIssues { get; set; }
}
```

DTOs (TitleVerification.API/DTOs):

DocumentUploadDto.cs:

```
public class DocumentUploadDto
{
    public string DocumentType { get; set; }
    public IFormFile File { get; set; }
}
```

Database Context (TitleVerification.API/Data):

AppDbContext.cs:

```
public class AppDbContext : DbContext
{
    public DbSet<Document> Documents { get; set; }

    public AppDbContext(DbContextOptions<AppDbContext> options) : base(options) { }
}
```

appsettings.json:

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=localhost;Database=TitleVerificationDB;Trusted_Connection=True;"
  },
  "Jwt": {
    "Key": "yourSecretKey",
    "Issuer": "yourIssuer",
    "Audience": "yourAudience"
  }
}
```

Program.cs Configuration:

```
var builder = WebApplication.CreateBuilder(args);
builder.Services.AddDbContext<AppDbContext>(options =>
    options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection")));
builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(options =>
    {
        var jwt = builder.Configuration.GetSection("Jwt");
        options.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuer = true,
            ValidateAudience = true,
            ValidateLifetime = true,
            ValidateIssuerSigningKey = true,
            ValidIssuer = jwt["Issuer"],
            ValidAudience = jwt["Audience"],
            IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(jwt["Key"]))
        };
    });
builder.Services.AddControllers();
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();
var app = builder.Build();
app.UseSwagger();
app.UseSwaggerUI();
app.UseAuthentication();
app.UseAuthorization();
app.MapControllers();
app.Run();
```

Controllers (TitleVerification.API/Controllers):

AuthController.cs:

```
[HttpPost("login")]
public IActionResult Login([FromBody] LoginDto dto)
{
    if (dto.Username == "admin" && dto.Password == "password")
    {
        var claims = new[]
        {
            new Claim(ClaimTypes.Name, dto.Username),
            new Claim(ClaimTypes.Role, "Admin")
        };
        var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_config["Jwt:Key"]));
        var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);
        var token = new JwtSecurityToken(
            issuer: _config["Jwt:Issuer"],
            audience: _config["Jwt:Audience"],
            claims: claims,
            expires: DateTime.Now.AddHours(1),
            signingCredentials: creds);
        return Ok(new { token = new JwtSecurityTokenHandler().WriteToken(token) });
    }
    return Unauthorized();
}
```

DocumentController.cs:

```
[HttpPost("upload")]
public async Task<IActionResult> Upload([FromForm] DocumentUploadDto dto)
{
    if (dto.File.Length > 10 * 1024 * 1024)
        return BadRequest("File size exceeds 10MB.");
    var ext = Path.GetExtension(dto.File.FileName).ToLower();
    if (ext != ".pdf" && ext != ".jpg" && ext != ".png")
        return BadRequest("Invalid file type.");
    var uploadsPath = Path.Combine(_env.ContentRootPath, "Uploads");
    Directory.CreateDirectory(uploadsPath);
    var filePath = Path.Combine(uploadsPath, dto.File.FileName);
    using var stream = new FileStream(filePath, FileMode.Create);
    await dto.File.CopyToAsync(stream);
    return Ok(new { message = "File uploaded successfully." });
}
```