Here’s a very detailed, step-by-step guide to setting up and starting the Hospital Management System (HMS) project using C# for the desktop app, Flutter for Android, and SQL Server as the database. Each step will focus on providing the smoothest setup experience possible.

Project Overview and Requirements

1. Objective: Build a HMS where a C# desktop app and Flutter mobile app can interact with a centralized backend API, using SQL Server as the primary database.
2. Technologies:

Desktop App: C# with WPF (Windows Presentation Foundation) for Windows.

Mobile App: Flutter for Android.

Backend API: .NET Core Web API for central data handling.

Database: SQL Server for structured data storage.

1. Development Environment:

Visual Studio for C# and .NET Core API.

Visual Studio Code (optional) or Android Studio for Flutter.

SQL Server Management Studio (SSMS) for managing SQL Server.

Step 1: Set Up SQL Server Database

1. Install SQL Server:

Download and install SQL Server (e.g., SQL Server 2019 or SQL Server Express).

Install SQL Server Management Studio (SSMS) to interact with the database.

1. Create the Database:

Open SSMS, connect to your SQL Server instance.

Right-click Databases > New Database….

Name your database, e.g., HMS\_Database, and click OK.

1. Define Database Schema:

Define tables for Users, Patients, Appointments, etc.

Here’s an example schema for some key tables:

CREATE TABLE Users (

UserId INT PRIMARY KEY IDENTITY,

Username NVARCHAR(50) UNIQUE NOT NULL,

PasswordHash NVARCHAR(256) NOT NULL,

Role NVARCHAR(20) NOT NULL

);

CREATE TABLE Patients (

PatientId INT PRIMARY KEY IDENTITY,

FirstName NVARCHAR(50),

LastName NVARCHAR(50),

DOB DATE,

MedicalHistory NVARCHAR(MAX)

);

CREATE TABLE Appointments (

AppointmentId INT PRIMARY KEY IDENTITY,

PatientId INT FOREIGN KEY REFERENCES Patients(PatientId),

DoctorId INT FOREIGN KEY REFERENCES Users(UserId),

DateTime DATETIME,

Status NVARCHAR(20)

);

1. Set Up Remote Connection (If Needed):

For remote access (e.g., from Flutter), ensure SQL Server is configured to allow remote connections.

In SQL Server Configuration Manager, enable TCP/IP under SQL Server Network Configuration.

Step 2: Create the Backend API with .NET Core

1. Install Visual Studio:

Install Visual Studio 2019 or 2022 with the .NET Core cross-platform development workload.

1. Create an ASP.NET Core Web API Project:

Open Visual Studio > Create a new project > Select ASP.NET Core Web API.

Name the project HMS\_API and set the Solution name to HMS\_Project.

Choose .NET 5 or .NET 6 as the target framework.

1. Install Entity Framework Core:

Open the NuGet Package Manager and install these packages:

Microsoft.EntityFrameworkCore.SqlServer

Microsoft.EntityFrameworkCore.Tools

1. Configure Database Connection:

Open appsettings.json and add your SQL Server connection string:

“ConnectionStrings”: {

“DefaultConnection”: “Server=YOUR\_SERVER\_NAME;Database=HMS\_Database;User Id=YOUR\_USERNAME;Password=YOUR\_PASSWORD;”

}

1. Add the Database Context:

Create a folder called Data, then add a new class called HMSDbContext.cs.

Using Microsoft.EntityFrameworkCore;

Public class HMSDbContext : DbContext

{

Public HMSDbContext(DbContextOptions<HMSDbContext> options) : base(options) { }

Public DbSet<User> Users { get; set; }

Public DbSet<Patient> Patients { get; set; }

Public DbSet<Appointment> Appointments { get; set; }

}

1. Add Models for Database Entities:

Create a folder called Models and define C# classes for each entity, e.g., User.cs, Patient.cs, Appointment.cs.

Public class User

{

Public int UserId { get; set; }

Public string Username { get; set; }

Public string PasswordHash { get; set; }

Public string Role { get; set; }

}

1. Configure Dependency Injection:

Open Startup.cs and register HMSDbContext in ConfigureServices:

Public void ConfigureServices(IServiceCollection services)

{

Services.AddDbContext<HMSDbContext>(options =>

Options.UseSqlServer(Configuration.GetConnectionString(“DefaultConnection”)));

Services.AddControllers();

}

1. Create API Controllers:

Create folders named Controllers and add controllers for Users, Patients, Appointments, etc.

[Route(“api/[controller]”)]

[ApiController]

Public class PatientController : ControllerBase

{

Private readonly HMSDbContext \_context;

Public PatientController(HMSDbContext context)

{

\_context = context;

}

[HttpGet(“{id}”)]

Public async Task<ActionResult<Patient>> GetPatient(int id)

{

Var patient = await \_context.Patients.FindAsync(id);

If (patient == null) return NotFound();

Return patient;

}

}

1. Run and Test the API:

Run the API by pressing F5 and test it using Postman to make sure endpoints work as expected.

Step 3: Set Up the Desktop App (C# WPF)

1. Create a WPF Project in Visual Studio:

Open Visual Studio > Create a new project > Select WPF App (.NET Core).

Name it HMS.DesktopApp.

1. Add an API Client:

Add System.Net.Http for making HTTP requests to the API.

1. Define Models and API Service:

Add a Models folder and create classes for User, Patient, etc., that match the backend models.

Create an ApiService.cs file to handle API requests.

Public class ApiService

{

Private readonly HttpClient \_client = new HttpClient { BaseAddress = new Uri(<https://localhost:5001/api/>) };

Public async Task<Patient> GetPatientAsync(int id)

{

Var response = await \_client.GetAsync($”patients/{id}”);

Response.EnsureSuccessStatusCode();

Return await response.Content.ReadAsAsync<Patient>();

}

}

1. Create UI and Call API:

Design WPF UI forms and connect buttons and data fields to make API calls.

Step 4: Set Up the Mobile App (Flutter)

1. Install Flutter:

Follow Flutter’s installation guide.

1. Create a New Flutter Project:

Open a terminal, navigate to your project directory, and create a new Flutter project:

Flutter create hms\_mobile

1. Add HTTP Package:

Open pubspec.yaml and add the http package under dependencies:

Dependencies:

http: ^0.14.0

1. Define Models and API Service:

Create a models folder and add Dart classes for entities like Patient, User.

Add an api\_service.dart to handle API requests.

Import ‘package:http/http.dart’ as http;

Class ApiService {

Final String baseUrl = ‘https://localhost:5001/api/’;

Future<Patient> getPatient(int id) async {

Final response = await http.get(Uri.parse(‘$baseUrl/patients/$id’));

If (response.statusCode == 200) {

Return Patient.fromJson(json.decode(response.body));

} else {

Throw Exception(‘Failed to load patient’);

}

}

}

1. Implement the UI:

Create screens like PatientList, AppointmentSchedule, and connect them to API data using FutureBuilder.

Step 5: Implement Authentication and Security

1. JWT Authentication:

Implement JWT authentication in the backend (e.g., add token generation in login endpoint).

Update the API service in both apps to add tokens to requests.

1. HTTPS Configuration:

Configure SSL certificates for HTTPS in development and deployment for secure communication.

Step 6: Deployment

1. Host the Backend API:

Deploy the API on a cloud server (e.g., Azure, AWS, or a VPS).

Set up SQL Server to allow remote connections.

1. Deploy Mobile and Desktop Apps:

Desktop: Package the WPF app using an installer (e.g., Inno Setup).

Flutter: Build and release an APK for Android