Setting up a Hospital Management System (HMS) with C# for the desktop app, Flutter for Android, and SQL Server as the database involves several steps to ensure efficient communication between the desktop, mobile, and backend services. Here’s the complete setup:

1. Project Structure Overview

This project will have the following key components:

1. Desktop App: Built in C# using .NET for the desktop application.
2. Mobile App: Developed in Flutter for Android, consuming APIs from the backend.
3. Backend Server: A central server to handle API requests, data processing, and database interaction, using .NET Web API.
4. Database: SQL Server as the relational database.
5. Database Setup (SQL Server)
6. Install SQL Server:

Download and install SQL Server (any version, such as SQL Server 2019 or SQL Server Express for development).

Install SQL Server Management Studio (SSMS) to manage your database.

1. Database Design:

Design your HMS database schema with tables like Users, Patients, Doctors, Prescriptions, Appointments, Drones, etc.

Use primary keys, foreign keys, indexes, and constraints to enforce data integrity.

1. Sample Schema:

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. Database Connection:

Set up SQL Server to allow remote connections if you need access from other devices (like mobile apps).

Create a connection string that your C# desktop app and backend server can use to access the database.

1. Backend Server Setup (ASP.NET Core Web API)
2. Create ASP.NET Core Web API Project:

Open Visual Studio and create a new ASP.NET Core Web API project named HMS.API.

Add Entity Framework Core for database interaction by running:

Dotnet add package Microsoft.EntityFrameworkCore.SqlServer

Dotnet add package Microsoft.EntityFrameworkCore.Tools

1. Configure Database Context:

In your project, create a Data folder and add an HMSDbContext class to manage the SQL Server connection.

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 Connection String in appsettings.json:

“ConnectionStrings”: {

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

}

1. Configure Dependency Injection in Startup.cs:

Public void ConfigureServices(IServiceCollection services)

{

Services.AddDbContext<HMSDbContext>(options =>

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

Services.AddControllers();

}

1. Create API Controllers:

Create controllers for Users, Patients, Appointments, etc., with CRUD operations.

Example PatientController:

[ApiController]

[Route(“api/[controller]”)]

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. Test APIs:

Run the project and use a tool like Postman to test each API endpoint.

1. C# Desktop App Setup
2. Create Desktop Project:

Create a WPF project in Visual Studio named HMS.DesktopApp.

1. Add API Client:

Install System.Net.Http to make HTTP requests to the backend API:

Dotnet add package System.Net.Http

1. Create Models and API Service Classes:

Add models like Patient, User, and service classes to communicate with the API.

Public class ApiService

{

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

Public async Task<Patient> GetPatientAsync(int id)

{

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

Response.EnsureSuccessStatusCode();

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

}

}

1. Implement UI and API Calls:

Create user interface forms for functionalities (e.g., viewing patients, scheduling appointments).

Call API methods in the UI to display data.

1. Flutter Mobile App Setup
2. Create Flutter Project:

Create a new Flutter project named hms\_mobile:

Flutter create hms\_mobile

1. Add Dependencies:

Add http for API requests:

Dependencies:

http: ^0.14.0

1. Setup Models and API Services:

Create a models folder with Dart classes (e.g., patient.dart, appointment.dart).

Set up an API service to interact with the backend.

Import ‘dart:convert’;

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. UI Implementation:

Design screens in Flutter (e.g., patient list, appointment booking).

Use FutureBuilder to handle API calls and display data asynchronously.

1. Authentication and Security
2. JWT Authentication:

Implement JWT authentication in the backend.

Configure Flutter and C# apps to store tokens and include them in API requests.

1. HTTPS and Certificates:

Ensure the API server uses HTTPS for secure data transmission.

For local development, you may need to configure Flutter and the C# app to accept self-signed certificates.

1. Deployment Setup
2. Host the API:

Deploy the API server on a cloud provider (AWS, Azure, etc.) or a VPS for remote access.

Configure SQL Server to allow connections from the server.

1. Configure Environment Variables:

Store sensitive information (e.g., database credentials) as environment variables in the server.

1. Build and Deploy Mobile and Desktop Apps:

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

Flutter Mobile: Build the APK for Android deployment.

Project Structure

HMS\_Project/

├── HMS.API/ # ASP.NET Core Web API

│ ├── Controllers/ # API Controllers

│ ├── Models/ # Entity Models

│ └── Data/ # Database Context

├── HMS.DesktopApp/ # C# WPF Desktop App

│ ├── Views/ # WPF Views

│ └── Services/ # API Service Classes

├── hms\_mobile/ # Flutter Mobile App

│ ├── lib/

│ │ ├── models/ # Dart Models

│ │ └── screens/ # Flutter Screens

└── Database/ # SQL Server Database

This setup provides a complete framework for building, deploying, and managing an HMS with centralized resources accessible from both desktop and mobile apps. Let me know if you need further details on specific parts!