

**РАЗРАБОТКА ТЕСТОВЫХ НАБОРОВ**

### **Тестовые наборы для проверки даты записи клиента**

#### **Назначение: Проверка корректности ввода даты**

1. Граничные значения

01.01.1900: Проверка нижней границы допустимого диапазона дат.

31.12.2100: Проверка верхней границы допустимого диапазона дат.

1. Високосные года

29.02.2020: Проверка на високосный год (дата существует).

29.02.2021: Проверка на невисокосный год (дата не существует).

1. Неправильный формат

31-12-2020: Проверка неправильного формата даты (должен быть DD.MM.YYYY).

2020.12.31: Проверка неправильного формата даты (должен быть DD.MM.YYYY).

1. Не существующие даты

30.02.2020: Проверка несуществующей даты в феврале.

31.04.2020: Проверка несуществующей даты в апреле.

1. Пустое значение

Пустое поле: Проверка реакции системы на пустое значение.

| Тестовый набор 1.01  Назначение: Проверка нижней границы допустимого диапазона дат  Дата: 01.01.1900  Ожидаемый результат: Система принимает дату  Тестовый набор 1.02  Назначение: Проверка верхней границы допустимого диапазона дат  Дата: 31.12.2100  Ожидаемый результат: Система принимает дату  Тестовый набор 1.03  Назначение: Проверка на високосный год (дата существует)  Дата: 29.02.2020  Ожидаемый результат: Система принимает дату  Тестовый набор 1.04  Назначение: Проверка на невисокосный год (дата не существует)  Дата: 29.02.2021  Ожидаемый результат: Система отклоняет дату  Тестовый набор 1.05  Назначение: Проверка неправильного формата даты  Дата: 31-12-2020  Ожидаемый результат: Система отклоняет дату |
| --- |

### **Тестовые наборы для проверки номера талона**

#### **Назначение: Проверка корректности ввода номера талона**

1. Граничные значения

0000000: Проверка минимального допустимого значения (если 0 допустим).

9999999: Проверка максимального допустимого значения.

1. Формат номера

0123456: Проверка номера, начинающегося с нуля.

1234567: Проверка стандартного формата номера.

1. Неправильный формат

12345: Проверка слишком короткого номера.

123456789: Проверка слишком длинного номера.

12A3456: Проверка наличия недопустимых символов (буквы).

1. Пустое значение

Пустое поле: Проверка реакции системы на пустое значение.

| Тестовый набор 2.01  Назначение: Проверка минимального допустимого значения  Номер талона: 0000000  Ожидаемый результат: Система принимает номер  Тестовый набор 2.02  Назначение: Проверка максимального допустимого значения  Номер талона: 9999999  Ожидаемый результат: Система принимает номер  Тестовый набор 2.03  Назначение: Проверка номера, начинающегося с нуля  Номер талона: 0123456  Ожидаемый результат: Система принимает номер  Тестовый набор 2.04  Назначение: Проверка слишком короткого номера  Номер талона: 12345  Ожидаемый результат: Система отклоняет номер  Тестовый набор 2.05  Назначение: Проверка слишком длинного номера  Номер талона: 123456789  Ожидаемый результат: Система отклоняет номер  Тестовый набор 2.06  Назначение: Проверка наличия недопустимых символов  Номер талона: 12A3456  Ожидаемый результат: Система отклоняет номер |
| --- |

**РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

#### **Оглавление**

1. Введение
2. Регистрация и авторизация
3. Просмотр доступных врачей и их расписания
4. Запись на прием к врачу
5. Отмена или перенос приема
6. Просмотр медицинского расписания и результатов
7. Получение оповещений
8. Рабочий график для врачей
9. Принятие/отклонение записей на прием
10. Управление базой данных (для администраторов)
11. Создание и редактирование учетных записей (для администраторов)
12. Управление услугами и расписанием (для администраторов)
13. Генерация отчетов и статистики (для администраторов)

### **1. Введение**

Приложение "Поликлиника" предназначено для удобного и эффективного взаимодействия пациентов, врачей и администраторов поликлиники. Оно предоставляет возможность записи на приемы, просмотра медицинских записей, получения информации о врачах и услугах, а также управления рабочим графиком и базой данных.

### **2. Регистрация и авторизация**

Для начала использования приложения необходимо пройти процесс регистрации и авторизации:

1. Откройте приложение "Поликлиника".
2. Нажмите кнопку "Регистрация".
3. Введите свои персональные данные: имя, электронную почту и пароль.
4. Следуйте инструкциям на экране для завершения регистрации.
5. После успешной регистрации войдите в свою учетную запись, используя свою электронную почту и пароль.

### **3. Просмотр доступных врачей и их расписания**

Для поиска и просмотра информации о врачах и их расписании:

1. После авторизации откройте раздел "Поиск врачей" или "Список врачей".
2. Вам будет предоставлена информация о доступных врачах, их специализации и расписании работы.
3. Используйте фильтры для уточнения поиска по специализации, местоположению и доступности.

### **4. Запись на прием к врачу**

Для записи на прием к конкретному врачу:

1. Выберите врача из списка доступных врачей.
2. Просмотрите его расписание.
3. Выберите удобное для вас время и дату приема.
4. Подтвердите запись на прием и дождитесь подтверждения от врача или системы.

### **5. Отмена или перенос приема**

Для отмены или переноса приема:

1. Откройте свое медицинское расписание в приложении.
2. Найдите прием, который вы хотите отменить или перенести.
3. Выберите опцию "Отменить" или "Перенести".
4. Подтвердите свое решение и дождитесь подтверждения от врача или системы.

### **6. Просмотр медицинского расписания и результатов**

Для просмотра своего медицинского расписания и результатов:

1. Откройте свой личный кабинет или раздел "Медицинское расписание".
2. Выберите дату и просмотрите свои назначения на приемы и результаты.
3. Получите доступ к медицинским записям, диагнозам и рекомендациям от врачей.

### **7. Получение оповещений**

Для получения оповещений о предстоящих приемах:

1. Перейдите в настройки приложения или профиль пользователя.
2. Включите оповещения о предстоящих приемах, удостоверьтесь, что приложению предоставлен доступ к интернету и разрешения для отправки оповещений.

### **8. Рабочий график для врачей**

Для просмотра своего рабочего графика:

1. Войдите в личный кабинет врача или раздел "Рабочий график".
2. Просмотрите свое рабочее расписание по дням и времени.

### **9. Принятие/отклонение записей на прием**

Для врачей, для принятия или отклонения записей на прием:

1. Откройте раздел "Приемы пациентов" или "Записи на прием".
2. Выберите запись, которую вы хотите принять или отклонить.
3. Нажмите кнопку "Принять" или "Отклонить" соответственно.
4. Подтвердите свое решение и информируйте пациента, если необходимо.

### **10. Управление базой данных (для администраторов)**

Для администраторов поликлиники доступны следующие функции:

* Управление базой данных врачей и пациентов.
* Создание и редактирование учетных записей врачей и пациентов.
* Управление доступными услугами и расписанием врачей.
* Генерация отчетов и статистики.

### **Заключение**

Это руководство поможет вам освоить основные функции приложения "Поликлиника". Если у вас возникли вопросы или проблемы, обратитесь к администратору или поддержке приложения.

**МОДУЛЬ 2 БД**

**Создание базы данных  
Откройте SQL Server Management Studio и подключитесь к серверу баз данных. Затем выполните следующий SQL-запрос для создания новой базы данных:**

**sql  
Копировать код  
CREATE DATABASE Polyclinic;**

**GO**

**Создание таблиц  
Теперь создадим таблицы для основных сущностей:  
Таблица Patients (Пациенты)  
sql  
Копировать код  
USE Polyclinic;**

**GO**

**CREATE TABLE Patients (**

**PatientID INT PRIMARY KEY IDENTITY(1,1),**

**FirstName NVARCHAR(50) NOT NULL,**

**LastName NVARCHAR(50) NOT NULL,**

**Email NVARCHAR(100) NOT NULL,**

**Phone NVARCHAR(20),**

**DateOfBirth DATE,**

**RegistrationDate DATETIME DEFAULT GETDATE()**

**);**

**GO**

**-- Вставляем тестовые данные**

**INSERT INTO Patients (FirstName, LastName, Email, Phone, DateOfBirth)**

**VALUES**

**('Иван', 'Иванов', 'ivan@example.com', '123-456-7890', '1980-01-15'),**

**('Мария', 'Петрова', 'maria@example.com', '234-567-8901', '1995-05-20'),**

**('Петр', 'Сидоров', 'petr@example.com', '345-678-9012', '1988-09-10'),**

**('Анна', 'Козлова', 'anna@example.com', '456-789-0123', '1975-03-25'),**

**('Сергей', 'Федоров', 'sergey@example.com', '567-890-1234', '1992-11-30');**

**GO**

**Таблица Doctors (Врачи)  
sql  
Копировать код  
CREATE TABLE Doctors (**

**DoctorID INT PRIMARY KEY IDENTITY(1,1),**

**FirstName NVARCHAR(50) NOT NULL,**

**LastName NVARCHAR(50) NOT NULL,**

**Specialty NVARCHAR(100) NOT NULL,**

**Email NVARCHAR(100) NOT NULL,**

**Phone NVARCHAR(20),**

**RegistrationDate DATETIME DEFAULT GETDATE()**

**);**

**GO**

**-- Вставляем тестовые данные**

**INSERT INTO Doctors (FirstName, LastName, Specialty, Email, Phone)**

**VALUES**

**('Алексей', 'Петров', 'Терапевт', 'alexey@example.com', '123-456-7890'),**

**('Елена', 'Смирнова', 'Стоматолог', 'elena@example.com', '234-567-8901'),**

**('Иван', 'Козлов', 'Хирург', 'ivan@example.com', '345-678-9012'),**

**('Наталья', 'Иванова', 'Окулист', 'natalia@example.com', '456-789-0123'),**

**('Сергей', 'Сидоров', 'Педиатр', 'sergey@example.com', '567-890-1234');**

**GO**

**Таблица Appointments (Записи на прием)  
sql  
Копировать код  
CREATE TABLE Appointments (**

**AppointmentID INT PRIMARY KEY IDENTITY(1,1),**

**PatientID INT NOT NULL,**

**DoctorID INT NOT NULL,**

**AppointmentDateTime DATETIME NOT NULL,**

**AppointmentType NVARCHAR(50),**

**Status NVARCHAR(50) DEFAULT 'Запланировано',**

**CONSTRAINT FK\_PatientID FOREIGN KEY (PatientID) REFERENCES Patients(PatientID),**

**CONSTRAINT FK\_DoctorID FOREIGN KEY (DoctorID) REFERENCES Doctors(DoctorID)**

**);**

**GO**

**-- Вставляем тестовые данные**

**INSERT INTO Appointments (PatientID, DoctorID, AppointmentDateTime, AppointmentType)**

**VALUES**

**(1, 1, '2024-06-25 10:00:00', 'Плановый осмотр'),**

**(2, 3, '2024-06-26 14:30:00', 'Лечение зубов'),**

**(3, 2, '2024-06-27 11:15:00', 'Консультация'),**

**(4, 4, '2024-06-28 09:00:00', 'Диагностика зрения'),**

**(5, 5, '2024-06-29 15:00:00', 'Педиатрический осмотр');**

**GO**

**Таблица MedicalRecords (Медицинские записи)  
sql  
Копировать код  
CREATE TABLE MedicalRecords (**

**RecordID INT PRIMARY KEY IDENTITY(1,1),**

**PatientID INT NOT NULL,**

**DoctorID INT NOT NULL,**

**RecordDate DATE NOT NULL,**

**Diagnosis NVARCHAR(200),**

**Prescription NVARCHAR(200),**

**CONSTRAINT FK\_PatientID\_MR FOREIGN KEY (PatientID) REFERENCES Patients(PatientID),**

**CONSTRAINT FK\_DoctorID\_MR FOREIGN KEY (DoctorID) REFERENCES Doctors(DoctorID)**

**);**

**GO**

**-- Вставляем тестовые данные**

**INSERT INTO MedicalRecords (PatientID, DoctorID, RecordDate, Diagnosis, Prescription)**

**VALUES**

**(1, 1, '2024-06-20', 'Гипертония', 'Прописаны таблетки'),**

**(2, 3, '2024-06-21', 'Кариес', 'Рекомендованы профилактические процедуры'),**

**(3, 2, '2024-06-22', 'Гастрит', 'Назначена диета и лекарства'),**

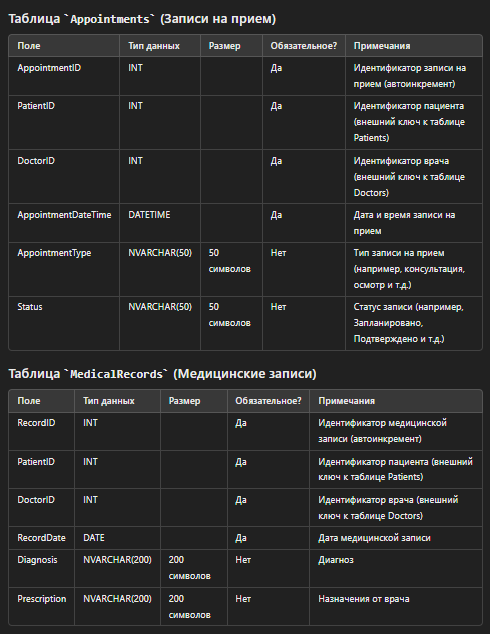
**(4, 4, '2024-06-23', 'Конъюнктивит', 'Рекомендованы капли для глаз'),**

**(5, 5, '2024-06-24', 'Простуда', 'Рекомендованы отдых и противовирусные препараты');**

**GO**

**2. Разработайте словарь данных для базы данных, указавпокаждойсущности данные:**

****



4. Создайте процедуру, которая проверяет данные в базеданныхнакорректность даты приема. Процедура должна выводить проверенныеданныес указанием соответствующего признака валидности (1-валидный, 0–невалидный).

CREATE PROCEDURE CheckAppointmentDateValidity

AS

BEGIN

SET NOCOUNT ON;

DECLARE @ValidAppointments TABLE (

AppointmentID INT,

AppointmentDateTime DATETIME,

IsValid BIT

);

-- Вставляем валидные записи на прием (дата приема больше текущей даты)

INSERT INTO @ValidAppointments (AppointmentID, AppointmentDateTime, IsValid)

SELECT AppointmentID, AppointmentDateTime, 1

FROM Appointments

WHERE AppointmentDateTime > GETDATE();

-- Вставляем невалидные записи на прием (дата приема меньше или равна текущей дате)

INSERT INTO @ValidAppointments (AppointmentID, AppointmentDateTime, IsValid)

SELECT AppointmentID, AppointmentDateTime, 0

FROM Appointments

WHERE AppointmentDateTime <= GETDATE();

-- Выводим результаты проверки

SELECT \* FROM @ValidAppointments;

END;

GO

5. Создайте триггер, который срабатывает при измененииданныхвтаблице приема пациентов и производит

-- Создание триггера для журнала изменений приемов пациентов

CREATE TRIGGER trg\_Appointments\_LogChanges

ON Appointments

AFTER INSERT, UPDATE, DELETE

AS

BEGIN

SET NOCOUNT ON;

DECLARE @LogMessage NVARCHAR(MAX);

IF EXISTS (SELECT \* FROM inserted)

BEGIN

-- Добавление записи о вставке или обновлении

INSERT INTO AppointmentChangesLog (ActionType, AppointmentID, AppointmentDateTime, ChangedDateTime)

SELECT

CASE

WHEN i.AppointmentID IS NOT NULL AND d.AppointmentID IS NOT NULL THEN 'Update' -- Обновление записи

WHEN i.AppointmentID IS NOT NULL THEN 'Insert' -- Вставка новой записи

END,

COALESCE(i.AppointmentID, d.AppointmentID),

COALESCE(i.AppointmentDateTime, d.AppointmentDateTime),

GETDATE()

FROM inserted i

FULL OUTER JOIN deleted d ON i.AppointmentID = d.AppointmentID

WHERE i.AppointmentID IS NOT NULL OR d.AppointmentID IS NOT NULL;

END

ELSE IF EXISTS (SELECT \* FROM deleted)

BEGIN

-- Добавление записи об удалении

INSERT INTO AppointmentChangesLog (ActionType, AppointmentID, AppointmentDateTime, ChangedDateTime)

SELECT 'Delete', d.AppointmentID, d.AppointmentDateTime, GETDATE()

FROM deleted d;

END

END;

GO

Создайте скрипт и бэкап базы данных, которыйпозволяетпровести резервное копирование базы данных.

BACKUP DATABASE ClinicDB

TO DISK = 'C:\Backup\ClinicDB.bak'

WITH FORMAT, INIT, COMPRESSION;

**Разработка API**

{ "ConnectionStrings": { "DefaultConnection": "Server=<server\_name>;Database=<database\_name>;User=<username>;Password=<password>;"

**Настройка сервисов и контекста базы данных в Startup.cs:**В методе ConfigureServices класса Startup.cs добавьте настройку сервисов и контекста базы данных. Ниже приведен пример для подключения к базе данных SQL Server с использованием Entity Framework Core.  
csharp  
Копировать код  
using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.DependencyInjection;

using YourNamespace.Data; // замените на ваше пространство имен

public class Startup

{

public Startup(IConfiguration configuration)

{

Configuration = configuration;

}

public IConfiguration Configuration { get; }

public void ConfigureServices(IServiceCollection services)

{

services.AddDbContext<ApplicationDbContext>(options =>

options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

// Добавьте другие сервисы, если необходимо

}

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

// Конфигурация приложения, например, middleware и обработка ошибок

}

}

* + ApplicationDbContext здесь представляет класс контекста вашей базы данных, который наследуется от DbContext и должен быть создан в вашем проекте.

**Создание класса контекста базы данных:**Пример класса контекста, который настроен для подключения к базе данных SQL Server с использованием Entity Framework Core:  
csharp  
Копировать код  
using Microsoft.EntityFrameworkCore;

namespace YourNamespace.Data // замените на ваше пространство имен

{

public class ApplicationDbContext : DbContext

{

public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) : base(options)

{

}

// DbSet для каждой таблицы в базе данных

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

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

// Добавьте другие DbSet, если необходимо

}

}

1. Здесь Patient и Appointment представляют модели данных, которые отображают таблицы в вашей базе данных.

**Использование контекста в контроллерах:**В ваших контроллерах API вы можете использовать контекст базы данных для выполнения операций CRUD. Например:  
csharp  
Копировать код  
using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using System.Collections.Generic;

using System.Threading.Tasks;

using YourNamespace.Data; // замените на ваше пространство имен

namespace ClinicManagementAPI.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class PatientsController : ControllerBase

{

private readonly ApplicationDbContext \_context;

public PatientsController(ApplicationDbContext context)

{

\_context = context;

}

// Пример метода для получения всех пациентов

[HttpGet]

public async Task<ActionResult<IEnumerable<Patient>>> GetPatients()

{

return await \_context.Patients.ToListAsync();

}

// Добавьте другие методы для CRUD операций

}

}

Эти шаги помогут вам настроить проект ASP.NET Core Web API для работы с базой данных. Убедитесь, что вы правильно установили пакеты NuGet для Entity Framework Core и любые другие необходимые зависимости.

**Создание моделей данных:**

* **Определите классы моделей данных, которые будут использоваться в вашем API. В данном случае, это классы Patient и Appointment, которые будут представлять сущности "Пациент" и "Запись на прием" соответственно.**

**Пример для Patient:  
csharp  
Копировать код  
public class Patient**

**{**

**public int Id { get; set; }**

**public string FirstName { get; set; }**

**public string LastName { get; set; }**

**public DateTime DateOfBirth { get; set; }**

**public string Gender { get; set; }**

**public string PhoneNumber { get; set; }**

**public string Email { get; set; }**

**public string Address { get; set; }**

**}**

**Пример для Appointment:  
csharp  
Копировать код  
public class Appointment**

**{**

**public int Id { get; set; }**

**public int PatientId { get; set; }**

**public int DoctorId { get; set; }**

**public DateTime AppointmentDateTime { get; set; }**

**public string ProblemDescription { get; set; }**

**}**

**Добавление моделей в проект:**

* **Создайте новую папку (например, "Models") в проекте и добавьте туда файлы с моделями данных.**

### **Шаг 3: Добавление контроллеров**

1. **Создание контроллеров:**
   * **Добавьте контроллеры для управления пациентами и записями на прием.**

**Пример контроллера для пациентов:  
csharp  
Копировать код  
[Route("api/[controller]")]**

**[ApiController]**

**public class PatientsController : ControllerBase**

**{**

**private readonly IPatientRepository \_patientRepository;**

**public PatientsController(IPatientRepository patientRepository)**

**{**

**\_patientRepository = patientRepository;**

**}**

**[HttpPost]**

**public ActionResult<int> CreatePatient(Patient patient)**

**{**

**// Реализация создания нового пациента**

**}**

**[HttpGet]**

**public ActionResult<IEnumerable<Patient>> GetPatients()**

**{**

**// Реализация получения списка всех пациентов**

**}**

**[HttpGet("{id}")]**

**public ActionResult<Patient> GetPatient(int id)**

**{**

**// Реализация получения информации о конкретном пациенте**

**}**

**[HttpPut("{id}")]**

**public IActionResult UpdatePatient(int id, Patient patient)**

**{**

**// Реализация обновления информации о пациенте**

**}**

**[HttpDelete("{id}")]**

**public IActionResult DeletePatient(int id)**

**{**

**// Реализация удаления пациента**

**}**

**}**

**Пример контроллера для записей на прием:  
csharp  
Копировать код  
[Route("api/[controller]")]**

**[ApiController]**

**public class AppointmentsController : ControllerBase**

**{**

**private readonly IAppointmentRepository \_appointmentRepository;**

**public AppointmentsController(IAppointmentRepository appointmentRepository)**

**{**

**\_appointmentRepository = appointmentRepository;**

**}**

**[HttpPost]**

**public ActionResult<int> CreateAppointment(Appointment appointment)**

**{**

**// Реализация создания новой записи на прием**

**}**

**[HttpGet]**

**public ActionResult<IEnumerable<Appointment>> GetAppointments()**

**{**

**// Реализация получения списка всех записей на прием**

**}**

**[HttpGet("{id}")]**

**public ActionResult<Appointment> GetAppointment(int id)**

**{**

**// Реализация получения информации о конкретной записи на прием**

**}**

**[HttpPut("{id}")]**

**public IActionResult UpdateAppointment(int id, Appointment appointment)**

**{**

**// Реализация обновления информации о записи на прием**

**}**

**[HttpDelete("{id}")]**

**public IActionResult DeleteAppointment(int id)**

**{**

**// Реализация удаления записи на прием**

**}**

**}**

### **Шаг 4: Настройка маршрутизации и зависимостей**

#### **Маршрутизация**

**ASP.NET Core использует атрибуты маршрутизации для определения маршрутов к методам контроллера. Для каждого контроллера вы определяете базовый путь, к которому добавляются пути к действиям методов.**

**Пример для контроллера PatientsController:**

**csharp**

**Копировать код**

**[Route("api/[controller]")]**

**[ApiController]**

**public class PatientsController : ControllerBase**

**{**

**// Конструктор и методы контроллера**

**}**

**Здесь [Route("api/[controller]")] указывает, что все методы контроллера будут доступны по пути /api/patients. Методы контроллера будут обрабатывать запросы к /api/patients, /api/patients/{id}, и т.д.**

#### **Инъекция зависимостей**

**ASP.NET Core поддерживает внедрение зависимостей (Dependency Injection, DI), что позволяет легко управлять зависимостями сервисов в контроллерах.**

**Пример:**

**csharp**

**Копировать код**

**public class PatientsController : ControllerBase**

**{**

**private readonly IPatientRepository \_patientRepository;**

**public PatientsController(IPatientRepository patientRepository)**

**{**

**\_patientRepository = patientRepository;**

**}**

**// Методы контроллера**

**}**

**В этом примере IPatientRepository внедряется через конструктор контроллера. IPatientRepository представляет интерфейс для работы с данными пациентов, и его реализация будет использоваться для доступа к базе данных или другому источнику данных.**

### **Шаг 5: Добавление авторизации**

#### **Использование аутентификации и авторизации**

**Для защиты API от несанкционированного доступа рекомендуется использовать механизмы аутентификации и авторизации, такие как JWT (JSON Web Tokens).**

1. **Настройка аутентификации:**
   * **Добавьте аутентификацию в ваш проект ASP.NET Core. Это можно сделать с помощью middleware и сервисов аутентификации, предоставляемых ASP.NET Core.**

**Пример настройки JWT аутентификации:  
csharp  
Копировать код  
public void ConfigureServices(IServiceCollection services)**

**{**

**services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)**

**.AddJwtBearer(options =>**

**{**

**options.TokenValidationParameters = new TokenValidationParameters**

**{**

**ValidateIssuer = true,**

**ValidateAudience = true,**

**ValidateLifetime = true,**

**ValidateIssuerSigningKey = true,**

**ValidIssuer = Configuration["Jwt:Issuer"],**

**ValidAudience = Configuration["Jwt:Audience"],**

**IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(Configuration["Jwt:Key"]))**

**};**

**});**

**// Другие сервисы и настройки**

**}**

1. **Использование атрибутов авторизации:**
   * **Добавьте [Authorize] атрибуты к методам контроллеров или к классам контроллеров, чтобы требовать аутентификацию для доступа к методам API.**

**Пример использования [Authorize] в контроллере:  
csharp  
Копировать код  
[Route("api/[controller]")]**

**[ApiController]**

**[Authorize]**

**public class PatientsController : ControllerBase**

**{**

**// Методы контроллера, доступные только авторизованным пользователям**

**}**

### **Шаг 6: Тестирование API**

#### **Использование инструментов для тестирования**

1. **Postman:**
   * **Установите и запустите Postman на вашем компьютере.**
   * **Создайте новый запрос для каждого метода вашего API (POST, GET, PUT, DELETE).**
   * **Укажите URL запроса (например, http://localhost:5000/api/patients).**
   * **Добавьте необходимые параметры и заголовки запроса.**
   * **Отправьте запрос и проверьте ответ на корректность.**
2. **Swagger:**
   * **Добавьте и настройте Swagger для вашего проекта ASP.NET Core.**
   * **Swagger предоставляет интерфейс для документирования и тестирования вашего API.**
   * **Запустите проект и перейдите по URL Swagger (http://localhost:5000/swagger), чтобы просмотреть и протестировать методы API.**

### **Шаг 7: Сохранение в системе контроля версий**

#### **Использование системы контроля версий**

1. **Git:**
   * **Инициализируйте систему контроля версий Git в вашем проекте, если еще не сделали.**
   * **Добавьте все файлы проекта в индекс Git с помощью команды git add ..**
   * **Сделайте коммит изменений с описанием (например, git commit -m "Added PatientsController").**
   * **Отправьте изменения на удаленный репозиторий с помощью команды git push.**

**Эти шаги помогут вам разработать, тестировать и поддерживать API для управления приложением "Поликлиника" с использованием ASP.NET Core и C#.**

**Документация API**

### **Документация API для Поликлиники**

#### **1. Создание нового пациента**

* **Путь: /api/patients**
* **Методы HTTP: POST**
* **Описание: Создает нового пациента в базе данных поликлиники.**

**Тело запроса (JSON):  
json  
Копировать код  
{**

**"firstName": "string",**

**"lastName": "string",**

**"dateOfBirth": "string (yyyy-MM-dd)",**

**"gender": "string",**

**"phoneNumber": "string",**

**"email": "string",**

**"address": "string"**

**}**

* **Обязательные поля: firstName, lastName, dateOfBirth, gender, phoneNumber, email, address.**

#### **2. Получение списка всех пациентов**

* **Путь: /api/patients**
* **Методы HTTP: GET**
* **Описание: Возвращает список всех пациентов зарегистрированных в поликлинике.**

#### **3. Получение информации о конкретном пациенте**

* **Путь: /api/patients/{id}**
* **Методы HTTP: GET**
* **Описание: Возвращает информацию о конкретном пациенте по его идентификатору.**
* **Параметры пути: {id} - идентификатор пациента.**

#### **4. Обновление информации о пациенте**

* **Путь: /api/patients/{id}**
* **Методы HTTP: PUT**
* **Описание: Обновляет информацию о пациенте по его идентификатору.**

**Тело запроса (JSON):  
json  
Копировать код  
{**

**"firstName": "string",**

**"lastName": "string",**

**"dateOfBirth": "string (yyyy-MM-dd)",**

**"gender": "string",**

**"phoneNumber": "string",**

**"email": "string",**

**"address": "string"**

**}**

* **Обязательные поля: хотя бы одно из полей: firstName, lastName, dateOfBirth, gender, phoneNumber, email, address.**

#### **5. Удаление пациента**

* **Путь: /api/patients/{id}**
* **Методы HTTP: DELETE**
* **Описание: Удаляет пациента из базы данных по его идентификатору.**
* **Параметры пути: {id} - идентификатор пациента.**

#### **6. Создание новой записи на прием к врачу**

* **Путь: /api/appointments**
* **Методы HTTP: POST**
* **Описание: Создает новую запись на прием к врачу.**

**Тело запроса (JSON):  
json  
Копировать код  
{**

**"patientId": "integer",**

**"doctorId": "integer",**

**"appointmentDateTime": "string (yyyy-MM-ddTHH:mm:ss)",**

**"problemDescription": "string"**

**}**

* **Обязательные поля: patientId, doctorId, appointmentDateTime.**

#### **7. Получение списка всех записей на прием**

* **Путь: /api/appointments**
* **Методы HTTP: GET**
* **Описание: Возвращает список всех записей на прием к врачу.**

#### **8. Получение информации о конкретной записи на прием**

* **Путь: /api/appointments/{id}**
* **Методы HTTP: GET**
* **Описание: Возвращает информацию о конкретной записи на прием к врачу по её идентификатору.**
* **Параметры пути: {id} - идентификатор записи на прием.**

#### **9. Обновление информации о записи на прием**

* **Путь: /api/appointments/{id}**
* **Методы HTTP: PUT**
* **Описание: Обновляет информацию о записи на прием к врачу по её идентификатору.**

**Тело запроса (JSON):  
json  
Копировать код  
{**

**"patientId": "integer",**

**"doctorId": "integer",**

**"appointmentDateTime": "string (yyyy-MM-ddTHH:mm:ss)",**

**"problemDescription": "string"**

**}**

* **Обязательные поля: хотя бы одно из полей: patientId, doctorId, appointmentDateTime.**

#### **10. Удаление записи на прием**

* **Путь: /api/appointments/{id}**
* **Методы HTTP: DELETE**
* **Описание: Удаляет запись на прием к врачу по её идентификатору.**
* **Параметры пути: {id} - идентификатор записи на прием.**