Proyecto

Creación de Diagramas UML

Diagrama de Casos de Uso

```
@startuml
title Medical Appointment System - Use Case Diagram

actor "User" as User

rectangle "Medical Appointment System" {
  usecase "Register Patient" as UC1
  usecase "Register Doctor" as UC2
  usecase "Schedule Appointment" as UC3
  usecase "Cancel Appointment" as UC4
  usecase "Reschedule Appointment" as UC5
  usecase "List All Appointments" as UC6
  usecase "View Appointments by Patient" as UC7
  usecase "View Appointments by Doctor" as UC8
}

User --> UC1
User --> UC2
User --> UC3
User --> UC4
User --> UC5
User --> UC6
User --> UC7
User --> UC8

@enduml
```

Descripción:

- El actor principal es **User**, que representa a la persona que maneja el sistema.
- Las funcionalidades principales son:
  1. **Register Patient:** Registrar un nuevo paciente con sus datos personales.
  2. **Register Doctor:** Registrar un nuevo médico con su especialidad.
  3. **Schedule Appointment:** Crear una cita médica asignando un paciente, un doctor y una fecha.
  4. **Cancel Appointment:** Cambiar el estado de una cita existente a “Cancelada”.
  5. **Reschedule Appointment:** Modificar la fecha de una cita ya programada.
  6. **List All Appointments:** Consultar todas las citas registradas en el sistema.
  7. **View Appointments by Patient:** Consultar todas las citas asociadas a un paciente específico.
  8. **View Appointments by Doctor:** Consultar las citas asignadas a un médico específico.

Diagrama de Clases

```
@startuml
title Medical Appointment System - Class Diagram

class Person {
  + int Id
  + string FirstName
  + string LastName
  + string SecondLastName
  + string Document
  + string Phone
  + string Email
}
```

```
class Patient {
  + int Age
}

class Doctor {
  + string Specialty
}

class Appointment {
  + int Id
  + int PatientId
  + int DoctorId
  + DateTime Date
  + string Reason
  + string Status
  --
  + Patient Patient
  + Doctor Doctor
}

Person <|-- Patient
Person <|-- Doctor

Patient "1" -- "0..*" Appointment : >
Doctor "1" -- "0..*" Appointment : >
@enduml
```

Clases principales:

- **Person:** Clase base que concentra los datos comunes (nombre, apellido, segundo apellido, documento, Teléfono y Correo electrónico).
- **Patient:** Hereda de `Person` y agrega su (`Age`).
- **Doctor:** Hereda de `Person` y agrega su (`Specialty`).
- **Appointment:** Representa una cita médica. Contiene los identificadores de paciente y doctor, la fecha, el motivo y el estado
    - Tiene referencias directas (`Patient`, `Doctor`).

Relaciones:

- `Person` → `Patient` y `Doctor`: herencia (indicada con `<|--`).
- `Patient` "1" -- "0..*" `Appointment`: un paciente puede tener cero o muchas citas.
- `Doctor` "1" -- "0..*" `Appointment`: un doctor puede atender cero o muchas citas.

Diagrama Entidad Relación

```
@startuml
title Medical Appointment System - Database ER Diagram

entity "Patients" {
  * Id : int <<PK>>
  --
  FirstName : string
  LastName : string
  SecondLastName : string
  Document : string <<UQ>>
  Phone : string
  Email : string
    Age : int
}

entity "Doctors" {
  * Id : int <<PK>>
  --
  FirstName : string
  LastName : string
  SecondLastName : string
  Document : string <<UQ>>
  Phone : string
  Email : string
  Specialty : string
}

entity "Appointments" {
  * Id : int <<PK>>
  --
```

```
  PatientId : int <<FK>>
  DoctorId : int <<FK>>
  Date : DateTime
  Reason : string
  Status : string
}

Patients ||--o{ Appointments : >
Doctors ||--o{ Appointments : >
@enduml
```

Entidades:

- **Patients:** almacena los datos de los pacientes.
- **Doctors:** almacena los datos de los doctores.
- **Appointments:** almacena las citas, relacionando pacientes y doctores mediante `PatientId` y `DoctorId`.

Relaciones:

- Un paciente puede tener muchas citas (`Patients ||--o{ Appointments`).
- Un doctor puede atender muchas citas (`Doctors ||--o{ Appointments`).

Estructura del Proyecto

```
MedicalAppointmentSystem/
│
├── Models/
│   ├── Person.cs
│   ├── Patient.cs
│   ├── Doctor.cs
│   └── Appointment.cs
│   → Definir las entidades.
│
├── Controllers/
│   ├── PatientController.cs
│   ├── DoctorController.cs
│   └── AppointmentController.cs
```

```
│   → Manejan las solicitudes HTTP (GET, POST, PUT, DELETE) y coordinan la lógica.
│
├── Views/
│   ├── Patients/
│   ├── Doctors/
│   └── Appointments/
│   → Contiene las vistas (HTML + C#) que renderizan la interfaz web.
│
├── Data/
│   └── AppDbContext.cs
│   → Maneja la conexión a la base de datos.
│
├── Migrations/
│     → Contiene las entidades mapeadas con Entity Framework.
│
├── Docs/
│   ├── UML_Diagrams/
│   ├── Database.sql
│   └── README.md
│   → Guarda tus diagramas, scripts SQL y documentación del proyecto.
│
└── Program.cs
   → Punto de entrada. Configura servicios (MVC, EF, conexión a BD) y ejecuta la app.
```

Creación de Los Modelos

```
public class Person
   {
      [Key]
      public int Id { get; set; }

      [Required, MaxLength(100)]
      public string FirstName { get; set; } = string.Empty;

      [Required, MaxLength(100)]
```

```
        public string LastName { get; set; } = string.Empty;

        [MaxLength(100)]
        public string? SecondLastName { get; set; }

        [Required, MaxLength(20)]
        public string Document { get; set; } = string.Empty;

        [MaxLength(20)]
        public string? Phone { get; set; }

        [MaxLength(100)]
        public string? Email { get; set;
    }
```

```
public class Patient : Person
    {
      [Required]
      public int Age { get; set; }
    }
```

```
public class Doctor : Person
    {
        [Required, MaxLength(100)]
        public string Specialty { get; set; } = string.Empty;
    }
```

```
public class Appointment
    {
        [Key]
        public int Id { get; set; }

        [Required]
        public int PatientId { get; set; }

        [Required]
```

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

    [Required]
    public DateTime Date { get; set; }

    [MaxLength(200)]
    public string? Reason { get; set; }

    [Required, MaxLength(20)]
    public string Status { get; set; } = "Scheduled";

    [ForeignKey(nameof(PatientId))]
    public Patient? Patient { get; set; }

    [ForeignKey(nameof(DoctorId))]
    public Doctor? Doctor { get; set; }
}
```