

**GESTÃO DE UM CENTRO DE SAÚDE – SEGUNDA FASE**  
**PROGRAMAÇÃO ORIENTADA AOS OBJETOS**

Eva Alexandra Pereira Gomes 27484

*Orientador*

Ernesto Casanova

dezembro, 2024

## Índice

1. Objetivo do sistema .....	3
2. Tecnologias Utilizadas .....	3
3. Estrutura do projeto .....	4
2.1. <i>Models</i> .....	4
2.2. <i>Utils</i> .....	5
2.3. <i>Forms</i> .....	5
3. Fluxo da Interface Gráfica .....	7
3.1. <i>ManagePatientsForm</i> .....	7
3.2. <i>ManageDoctorsForm</i> .....	8
3.3. Formulário de Consultas ( <i>ManageAppointmentsForm</i> ) .....	8
4. Estrutura dos Arquivos JSON .....	9
5. Melhorias Futuras .....	11
6. Considerações finais .....	11

## Índice de Ilustrações

Figura 1: Representação da hierarquia de classes. ....	4
Figura 2: Lógica e relação entre os diferentes ficheiros.....	6

## 1. Objetivo do sistema

O sistema de Gestão de Clínica de Saúde desenvolvido em C# tem como objetivo facilitar a administração de uma clínica de saúde, incluindo a gestão de médicos e pacientes, medicação e exames prescritos pelos médicos, consultas e salas. O sistema permite o registo de pacientes, médicos e consultas. Os médicos podem indicar as medicações e exames realizados. O sistema inclui uma interface gráfica baseada em *Windows Forms*, com funcionalidades integradas para manipulação de dados utilizando serviços e persistência em arquivos JSON.

## 2. Tecnologias Utilizadas

- **Linguagem de Programação:** C#
- **Framework:** *Windows Forms*
- **Persistência de Dados:** Arquivos JSON
- **Ferramentas:** *Visual Studio*, *PlantUML* (para diagramas) e *Doxygen* (para documentação).

### 3. Estrutura do projeto

O projeto é uma aplicação para gestão de uma clínica de saúde (*HealthClinic*). Ele é dividido em três pastas principais:

1. **Models:** Representam as entidades principais da clínica (pessoas, pacientes, médicos, consultas, exames e medicamentos). Contêm as funcionalidades que manipulam os dados e executam ações sobre os modelos.
2. **Utils:** Fornecem métodos auxiliares, como validação de dados e persistência em arquivos.
3. **Forms:** Representam a interface gráfica (GUI) para os usuários interagirem com o sistema.

#### 2.1. Models

Constituídos pelos ficheiros/classes:

- **Person:** Classe base para *Patient* (Paciente) e *Doctor* (Médico). Inclui atributos comuns, como *Name*, *DateOfBirth*, *Gender* e *ContactInfo*.
- **Patient:** Herda de *Person* e adiciona os atributos *MedicalHistory*, *NIF* e *Adress*. Representa os pacientes da clínica.
- **Doctor:** Herda de *Person* e adiciona os atributos *Specialty* (especialidade) e *ConsultationFee* (custo da consulta). Representa os médicos.
- **Appointment:** Representa uma consulta médica, com referência a um paciente, um médico, sala (*OfficeRoom*), exames prescritos (*PrescribedExams*) e medicamentos (*PrescribedMedications*). Também calcula o custo total da consulta.

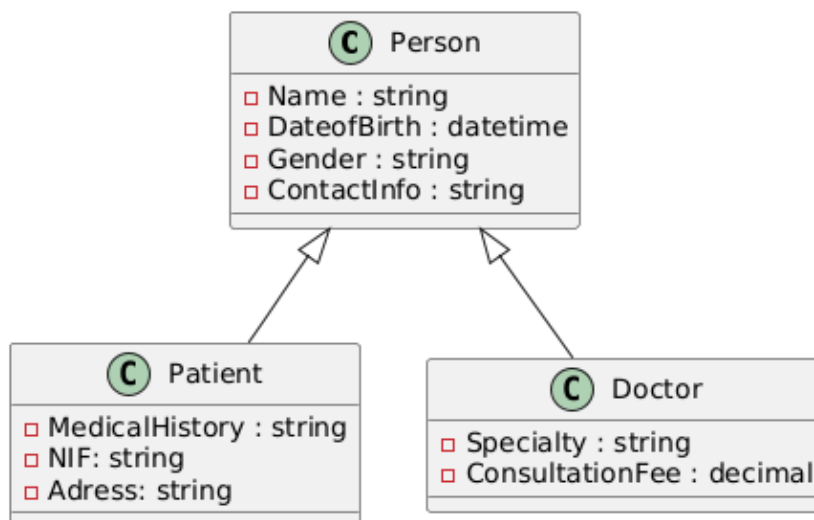


Figura 1: Representação da hierarquia de classes.

- **Exam e Medication:** Representam, respetivamente, exames e medicamentos, com atributos *Name* e *Cost*.
- **PatientServices:** Gere os pacientes, adiciona, remove, pesquisa e exibe todos os pacientes.

- **DoctorServices:** Gere os médicos, adiciona, remove, pesquisa e exibe todos os médicos.
- **AppointmentService:** Gere as consultas, cria novas consultas, obtém IDs únicos e exibe todas as consultas.

## 2.2. Utils

Os ficheiros utils fornecem suporte técnico ao projeto:

- **DataPersistence:** Salva e carrega os dados das entidades (*Patient*, *Doctor*, *Appointments*) em arquivos JSON.
- **DataValidation:** Valida a submissão de valores do usuário da plataformas, como números inteiros, decimais e *strings* não vazias.

Essas classes são reutilizáveis e ajudam a manter o código modular.

## 2.3. Forms

Os ficheiros forms são a camada de interface gráfica que permite aos usuários interagir com o sistema:

- **MainMenu:** Formulário principal que dá acesso às funcionalidades de gerenciamento de pacientes, médicos e consultas.
- **ManagePatientsForm:** Gere pacientes, com opções para adicionar, visualizar, buscar e remover.
- **ManageDoctorsForm:** Gere médicos, com funcionalidades semelhantes às dos pacientes.
- **ManageAppointmentsForm:** Gere consultas, permitindo adicionar, visualizar, buscar e remover consultas.

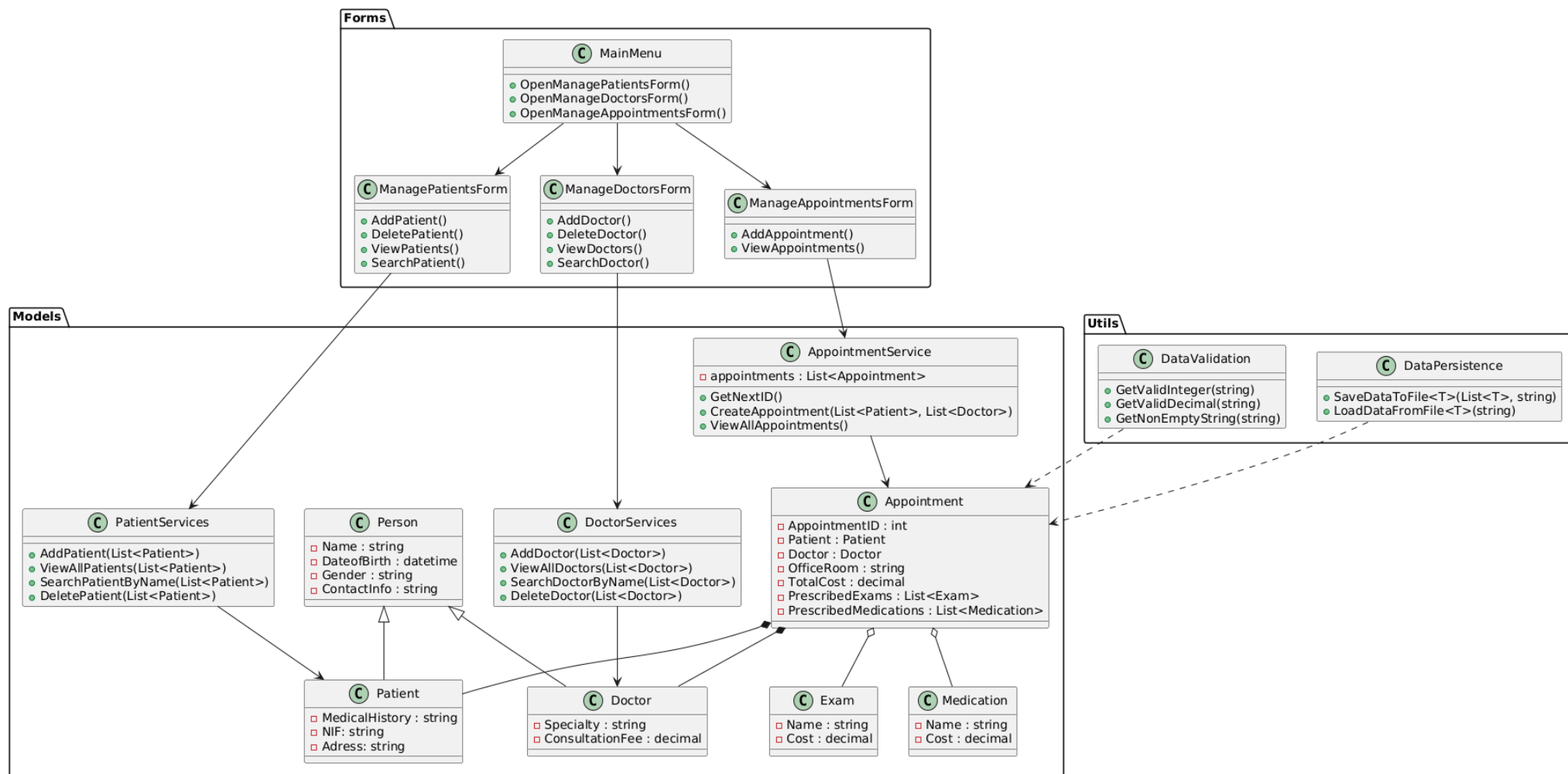


Figura 2: Lógica e relação entre os diferentes ficheiros.

### 3. Fluxo da Interface Gráfica

1. O usuário inicia o programa no formulário principal (**MainMenu**).

Para acessar ao sistema, use as seguintes credenciais:

- **Administrador:**
    - Utilizador: admin
    - Senha: password123
  - **Usuário Padrão:**
    - Utilizador: user1
    - Senha: userpassword
2. A partir do **MainMenu**, ele pode abrir:
    - O **ManagePatientsForm** para gerir pacientes.
    - O **ManageDoctorsForm** para gerir médicos.
    - O **ManageAppointmentsForm** para gerir consultas.

#### 3.1. *ManagePatientsForm*

**Operações possíveis:**

- **Adicionar paciente:**
  - O formulário adiciona os dados do paciente através de campos de entrada (*TextBox*, *ComboBox*, etc.).
  - Esses dados são enviados para o método `AddPatient()` do *PatientServices*.
  - O paciente é salvo num arquivo JSON usando *DataPersistence*.
- **Visualizar lista de pacientes:**
  - O formulário exibe todos os pacientes usando o método `ViewAllPatients()` do *PatientServices*.
  - Os pacientes carregados são exibidos numa tabela ou lista gráfica, como um *ListView* ou *DataGridView*.
- **Procurar paciente:**
  - O usuário pode digitar o nome do paciente num campo de busca.
  - O método `SearchPatientByName()` no *PatientServices* retorna o paciente correspondente.
  - Os detalhes do paciente são exibidos na interface.
- **Eliminar paciente:**
  - O usuário seleciona um paciente em uma lista ou tabela.
  - O método `DeletePatient()` no *PatientServices* remove o paciente da lista e atualiza o arquivo JSON.



### 3.2. *ManageDoctorsForm*

#### Operações possíveis:

- **Adicionar médico:**
  - Similar ao formulário de pacientes, o usuário preenche os detalhes do médico.
  - O método `AddDoctor()` no *DoctorServices* adiciona o médico.
  - Os dados são persistidos no arquivo JSON.
- **Visualizar lista de médicos:**
  - O método `ViewAllDoctors()` no *DoctorServices* carrega todos os médicos.
  - Os médicos são exibidos num componente gráfico, como *DataGridView*.
- **Procurar médico:**
  - O método `SearchDoctorByName()` no *DoctorServices* retorna o médico pelo nome.
  - O resultado é exibido em uma interface gráfica.
- **Eliminar médico:**
  - O método `DeleteDoctor()` no *DoctorServices* remove o médico da lista.
  - O arquivo JSON é atualizado após a exclusão.

### 3.3. *Formulário de Consultas (ManageAppointmentsForm)*

#### Operações possíveis:

- **Adicionar consulta:**
  - O formulário coleta:
    - O paciente e o médico (selecionados de listas pré-carregadas).
    - A sala de consulta (*OfficeRoom*).
    - Data e hora da consulta (*DateTimePicker*).
    - Exames e medicamentos prescritos.
  - Os dados são enviados ao método `CreateAppointment()` no *AppointmentService*, que:
    - Gera um novo ID para a consulta.
    - Calcula o custo total (consulta + exames + medicamentos).
    - Salva a consulta no arquivo JSON.
- **Visualizar consultas:**
  - O método `ViewAllAppointments()` no *AppointmentService* carrega todas as consultas.
  - As consultas são exibidas em uma tabela (*DataGridView*) com informações como:
    - Paciente, médico, sala, custo total, data e hora.
- **Detalhes da consulta:**
  - O formulário pode abrir um subformulário (*AppointmentDetailsForm*) para exibir detalhes completos de uma consulta:
    - Exames e medicamentos prescritos.
    - Paciente e médico associados.
    - Custo total e sala.

## 4. Estrutura dos Arquivos JSON

Os dados para as diferentes entidades (pacientes, médicos e consultas) são armazenados em arquivos JSON separados. Cada arquivo contém uma coleção de objetos representando as informações relevantes para a entidade correspondente.

### Exemplos:

- **patients.json:**

```
[
  {
    "$id": "2",
    "MedicalHistory": "diabetes",
    "NIF": "325245",
    "Address": "building A, Barcelos",
    "Name": "Marco Gomes",
    "Gender": "Male",
    "ContactInfo": "952352456",
    "DateOfBirth": "1984-12-21T00:00:00",
    "Age": 40
  }
]
```

- **doctors.json:**

```
[
  {
    "$id": "2",
    "Specialty": "surgery",
    "ConsultationFee": 190.0,
    "Name": "Susana Pereira",
    "Gender": "Female",
    "ContactInfo": "9532562",
    "DateOfBirth": "1999-12-21T00:00:00",
    "Age": 25
  }
]
```

- **appointments.json:**

```
[
  {
    "$id": "12",
    "AppointmentID": 3,
    "Patient": {
      "$id": "13",
      "MedicalHistory": "diabetes",
      "NIF": null,
      "Address": null,
      "Name": "Ana Soares",
      "Gender": "Female",
      "ContactInfo": "9133242",
      "DateOfBirth": "1999-01-04T00:00:00",
      "Age": 25
    },
    "Doctor": {
      "$id": "14",
      "Specialty": "pedologist",
      "ConsultationFee": 35.0,
      "Name": "Fernando",
      "Gender": "Female",
      "ContactInfo": "9108323",
      "DateOfBirth": "0001-01-01T00:00:00",
      "Age": 2023
    },
    "OfficeRoom": "3",
    "TotalCost": 37.0,
    "PrescribedExams": [
      {
        "$id": "15",
        "Name": "blood analysis",
        "Cost": 2.0
      }
    ],
    "PrescribedMedications": [],
    "AppointmentDate": "2024-12-21T19:25:59.0318686+00:00"
  }
]
```

## 5. Melhorias Futuras

- **Implementar banco de dados em vez de ficheiro JSON:** A substituição de arquivos JSON por um banco de dados oferece maior robustez, escalabilidade e segurança para o sistema *HealthClinic*.
- **Garantir maior proteção de dados.**
- **Permitir que os pacientes agendem consultas com médicos específicos:** Desenvolver uma funcionalidade onde os pacientes possam visualizar os horários disponíveis de cada médico e agendar consultas diretamente, reduzindo a carga administrativa e melhorando a experiência do paciente.
- **Desenvolver uma versão *web* ou *mobile* do sistema.**

## 6. Considerações finais

O sistema de gestão de consultas da Clínica de Saúde foi desenvolvido com o intuito de melhorar a eficiência e organização do atendimento a pacientes. Através da utilização de classes bem definidas e métodos para gerir as operações, o projeto proporciona uma base sólida para futuras expansões e melhorias.

Este projeto pode ser uma excelente base para o desenvolvimento de um sistema mais complexo e abrangente, que atenda a todas as necessidades de uma clínica moderna.