INSTITUTO POLITÉCNICO
DO CÁVADO E DO AVE
ESCOLA SUPERIOR DE TECNOLOGIA

RELATÓRIO DE TRABALHO PRÁTICO I

Sistema de Gestão em uma Crise de Saúde Pública

LUCAS BRGA MENDONÇA

ALUNO Nº 17870

Trabalho realizado sob a orientação de:
Luís Ferreira

Linguagens de Programação II

Licenciatura em Engenharia de Sistemas Informáticos

Barcelos, Maio de 2020

Índice

Lista de Figuras

1 Introdução

O presente trabalho tem como objetivo principal construir um sistema de gestão com os paradigmas das linguagens orientadas a objeto, especificamente o C#, que auxilie de alguma maneira numa crise de saúde pública. O sistema desenvolvido é capaz auxiliar na gestão de pessoas infectadas, tais como registro de novos casos, contabilização total de casos por região, sexo, faixa etária, dentre outros que são úteis no mesmo âmbito.

Para a primeira entrega do projeto, foram estruturadas as principais classes, assim como o relacionamento entre estas mesmas classes. Para a segunda entrega, novas classes foram criadas e o sistema estruturou-se tendo como base o padrão MVC com duas camadas adicionais: a de Serviços e a de Repositório. A camada de Serviço trabalha com as regras de negócio definidas para a execução de uma lógica pré-definida e customizável, é nas classes desta camada onde implementa-se as regras da gestão do sistema e quem tem acesso a esta camada é o controlador. Já a camada de Repositório funciona como uma camada intermediária entre o modelo de dados e o Serviço.

O código está estruturado de forma que as classes sejam fácilmente identificadas e de forma que utilize alguns dos conceitos fundamentais vistos até hoje nas aulas de Linguagem de Programação II. O presente relatório, tal como o código completo do projeto está disponível no GitHub, através do link 17870_LP2.git.

2 Estrutura do projeto e ferramentas auxiliares

Para uma melhor disposição e estruturação dos dados, as classes do programa foram divididas da seguinte maneira:

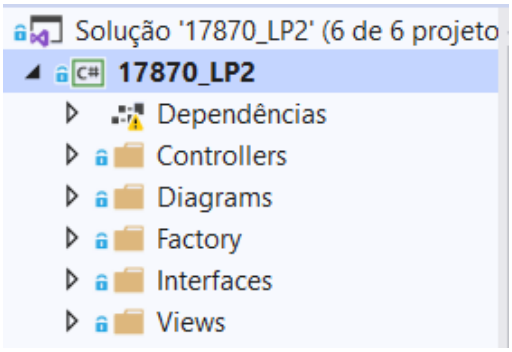

Figura 1: Pastas do projeto principal

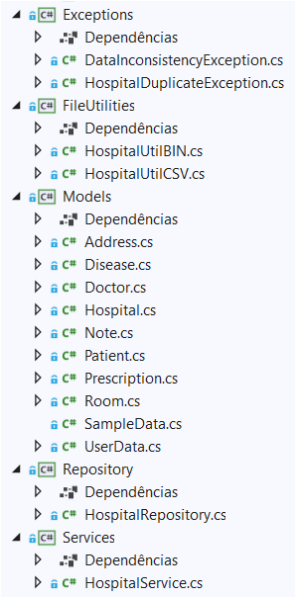

Figura 2: Bibliotecas criadas para o projeto

Controllers

Camada de Controle no modelo MVC. Contém a classe controladora principal do programa HospitalController.cs. É o componente que faz a mediação da entrada e saída de dados, comandando a visão e utilizando-se da classe de serviço adicional para relização de lógica para uma determinada regra de negócio. O foco do controlador é apenas ser um "intermediário" entre as várias camadas. Partindo do nosso tema em específico, os métodos que gerem uma lista de hospitais e adicionam características a estes hospitais implementados efetivamente na classe de Serviços.

Factory

Camada que contém as classes auxiliares para criar e gerenciar alguns tipos de objetos durante o programa. A classe Addresses.cs (plural), por exemplo, possui um atributo lista que contém objetos do tipo Address.cs (singular), assim como um método para a criação de um objeto Address e que adicionalmente fará parte da lista de objetos que a classe pode gerenciar.

Interfaces

Contém uma única interface denominada IHospitalView.cs que pode ser implementada em qualquer View do sistema. Possui dois métodos que devem ser implementados. O primeiro é o SetController(), método responsável por receber um controlador e associar este controlador a View. O segundo é o Display() onde efetivamente estará toda a lógica de programação para a apresentação e inserção de novos dados por parte do utilizador.

Views

Camada das Views no modelo MVC. Contém a view principal do programa até o momento. Configura-se como a saída de representação dos dados e possibilita o utilizador a inserção de outros. Implementa a interface IHospitalView.cs que possui os métodos descritos na linha (c).

Exceptions

Biblioteca que possui duas classes customizadas de possíveis exceptions que ocorram durante a execução do programa. Ainda, possuem uma lista estática como atributo para o gerenciamento das exceptions. Essa lista é salva como um ficheiro (BIN) de log de erros, com respectiva data e hora da ocorrência, assim como seu motivo.

FileUtilities

Biblioteca auxiliar para o gerenciamento de ficheiros. É responsável por guardar e recuperar os dados do sistema em ficheiros BIN ou CSV.

Models

Camada Model no modelo MVC. Biblioteca que contém as classes responsáveis por manter a estrutura dos dados da aplicação. Pode, num exemplo concreto, recuperar e armazenar o estado do modelo em um servidor de banco de dados. As classes aqui são utilizadas como base na criação de objetos.

Repository

Biblioteca que possui a classe responsável por gerir o modelo de dados, tais como salvar os dados ou recuperá-los utilizando as classes auxiliares de ficheiros.

Services

Biblioteca que contém a classe principal responsável por gerir a camada de negócios do sistema. Implementa os métodos associados a gestão de hospitais e seus atributos.

2.1 Associação entre os objetos

A ligação entre os objetos foi feita de forma a facilitar todo o processo de conexão entre os mais variados objetos dentro do sistema. Em suma, há um objeto Hospital, que por sua vez possui listas de atributos tais como pacientes (Patient.cs), quartos (Room.cs), doutores (Doctor.cs) e suas características principais. O diagrama de classes pode ser descrito detalhadamente abaixo:

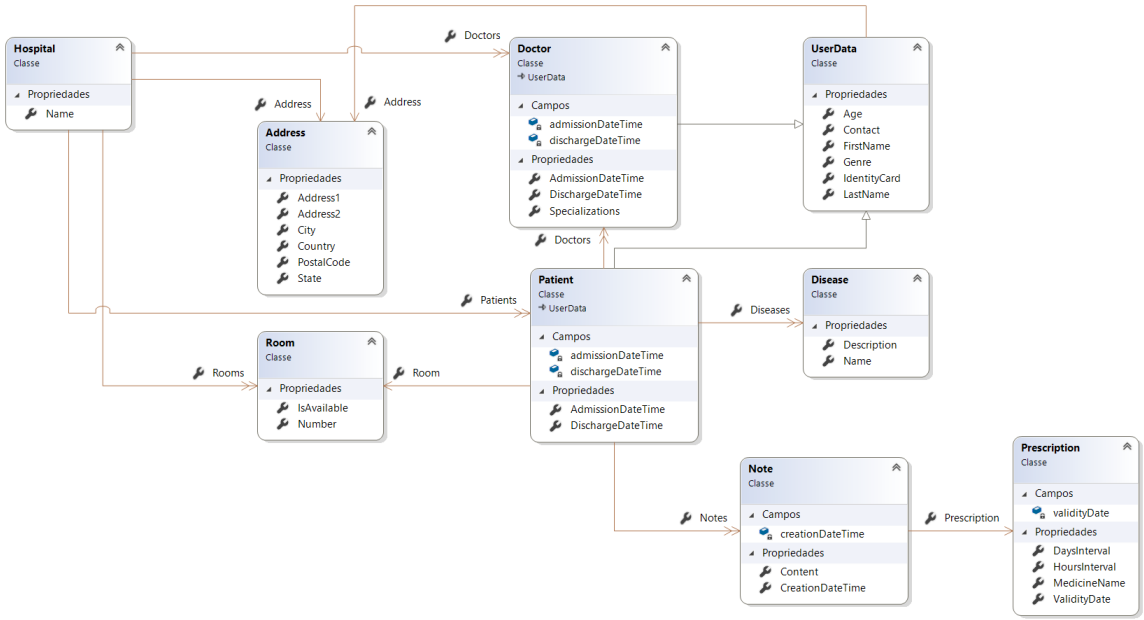

Figura 3: Diagrama de classes para o modelo de dados

Descrição das principais classes modelo:

UserData.cs

Classe responsável pelos dados de um usuário. É uma classe geral utilizada como "pai" de outras classes dentro do sistema.

Patient.cs

Classe responsável pelos dados de um paciente. Herda o modelo de dados de UserData.

Doctor.cs

Classe responsável pelos dados de um médico. Herda o modelo de dados de UserData.

Address.cs

Classe responsável pelos dados de endereços de usuários cadastrados no sistema, sejam eles pacientes ou médicos, além do modelo ser utilizado para endereços de hospitais.

Disease.cs

Classe responsável pelos dados de doenças associadas aos pacientes de um hospital.

Hospital.cs

Classe que tem dados de hospitais.

Note.cs

Classe quem tem os dados de um ficha médica de um paciente. É nesta classe que são encontradas informações relativas ao diagnóstico do doente, observações e preescrições médicas.

Prescription.cs

Classe responsável pelos dados de uma receita associada a uma nota (ficha médica). É nesta classe que estão dispostas informações sobre o medicamento receitado, assim como seu intervalo de dias e horas e prazo de válidade da mesma. Herda modelo de dados de PrescriptionData.cs.

Room.cs

Classe responsável por gerir os quartos disponíveis em um hospital.

Enumeração: Specialization.cs

Contém as especializações médicas disponíveis no sistema.

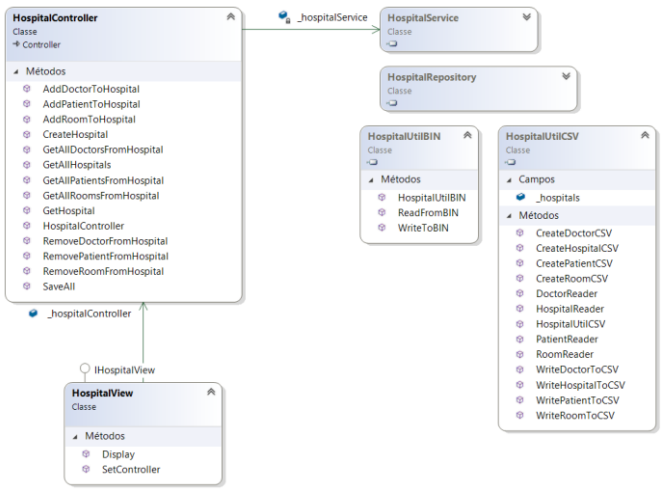

Figura 4: Diagrama de classes para o design pattern utilizado

O programa é inicializado através da classe Program.cs, que por sua vez inicializa o controlador principal do programa.

2.2 Uso de LINQ

A classe de serviço HospitalService.cs utiliza a biblioteca LINQ de forma a facilitar a consulta e a inserção de dados na lista de hospitais geridas pela mesma classe. LINQ (Language-Integrated Query) é um componente do Microsoft .NET que adiciona funcionalidades de consulta em fontes variadas.

```
//Checks if the patient has not been admitted to any other hospital
if(_hospitals.ToList().Any(h => h.Patients.Contains(patient))) {
        throw new DataInconsistencyException("Patient with identity Card " + patient.IdentityCard +
                        " is already in some Hospital ");
}

//Check if the room is already registered at the hospital
var findRoom = hospital.Rooms.Where(i => i.Number == room.Number).FirstOrDefault();
```

Figura 5: Exemplo de uso de LINQ

2.3 Carregamento lento: Entity Framework

Algumas propriedades de relacionamento foram marcadas como virtuais, pois é assim que o padrão Entity Framework pode implementar o chamado “carregamento lento” de forma quase que automática. Este carregamento significa que os dados relacionados são carregados de modo transparente do banco de dados quando a propriedade de navegação é acessada. Dado um contexto de acesso ao banco de dados, a propriedade que possui a relação só carregaria seus elementos se fosse realmente usada.

Embora o padrão Entity Framework e o carregamento lento não sejam utilizados no sistema até o momento, a escolha de seguir desenvolver de acordo com esses padrões é uma forma de praticar os conceitos até o cenário hipotético tornar-se realmente necessário.

3 Conclusão

Conclui-se que ao final da presente Entrega Número Dois, o sistema esteja estruturado em camadas de forma a facilitar a manutenção e reutilização de classes e bibliotecas. Se houvesse a necessidade de resolver algum problema relacionado ao carregamento de dados ao iniciar a aplicação e indicações pertinentes de que a leitura do ficheiro de dados não estivesse a funcionar da forma adequada, só haveria a necessidade de fazer modificações a biblioteca auxiliar *FileUtilities* que possui as classes auxiliares para carregar os dados. Da mesma forma que o sistema até então foi desenvolvido tendo como objeto de negócios principal uma lista de hospitais, que por sua vez possuem características associadas à outros objetos, como pacientes e doutores. Esses últimos deverão ser criados de forma isolada, onde cada classe de objeto deve ser responsável por gerir as características que lhe competem, não fazendo sentido a classe de Serviço que gere Hospitais criar endereços, por exemplo, apesar de não existir nada que impeça o programador de codificar de tal forma. Injetamos então, uma dependência na classe que gerencia os hospitais, e que espera como parâmetro os objetos já criados por outras classes, ficando responsável apenas por gerenciar a lista de hospitais, assim como fazer a associação entre um hospital e um paciente (dentro da mesma lista de hospitais).

Funcionalidades adicionais poderão ser criadas em versão posteriores para facilitar a consulta de certos dados estatísticos e a biblioteca LINQ pode ser peça fundamental, pelo menos a nível de execução de consulta a objetos e não a base de dados. Além da criação de novas views e novos controladores para ajudar na gestão do sistema como um todo. Em suma, a versão atual tenta entregar o que se propõe a nível de estrututuração de camadas e busca seguir um padrão de programação onde eventuais mudanças não comprometam o funcionamento do sistema. Entretanto, futuramente havendo a necessidade de mudar alguma lógica já implementada, implica-se que os objetos poderão ser alterados de forma isolada, sem impactos no funcionamento do sistema como um todo. Ainda, a documentação que descreve todo o processo de lógica de execução do programa está descrita juntamente com o código-fonte.