



Relatório

Linguagens de Programação II

Alunos:
José Dias (18840)
Pedro Ferraz (18842)

Professor: Luis Ferreira

Licenciatura em Engenharia de Sistemas Informáticos

Barcelos, abril, 2020

Resumo

Neste trabalho prático, estamos perante uma empresa que presta serviços de gestão e contabilidade e pretende proteger ao máximo a privacidade dos dados pessoais dos seus clientes, não colocando estes em qualquer perigo e aplicando as regras de acordo com o RGPD.

Para melhorar a proteção destes dados e a livre circulação deles, o departamento de informática da empresa pretende contratar um programador para desenvolver um programa capaz de auxiliar a manter o sistema informático da empresa em sintonia com o RGPD.

Aqui surge o trabalho prático, na criação de um código para ajudar os colaboradores responsáveis da empresa nas auditorias internas ao sistema informático, realizadas pelas autoridades competentes.

O código contém bastantes funcionalidades relacionadas com as auditorias, onde é posto em prática toda a matéria estudada durante a unidade curricular de Linguagens de Programação II.

Neste programa é possível analisar a ligação entre colaboradores, vulnerabilidades e equipamentos intervenientes numa auditoria. E ainda contem várias funcionalidades para remover e editar colaboradores e vulnerabilidades.

Link GitHub: https://github.com/josed14s/18840_18842_LP2

Lista de Abreviaturas e Siglas

- **RGPD** – Regulamento Geral de Proteção de Dados;
- **UE** – União Europeia;
- **DPO** – Data Protection Officer;

Índice de Figuras

Figura 1 - Simbolo Microsoft Visual Studio	5
Figura 2 - Regulamento Geral de Proteção de Dados	7
Figura 3 - diagrama de classes.....	14

Índice

1. Introdução	5
1.1. Contextualização	5
1.2. Motivação e objetivo.....	5
1.3. Estrutura do Documento.....	6
2. Estado da arte	7
3. Implementação	8
3.1. Descrição do problema	8
3.2. Solução	8
3.3. Program.cs.....	8
3.4. Auditoria.cs	8
3.5. Colaborador.cs	10
3.6. Vulnerabilidade.cs	11
3.7. Equipamento.cs.....	12
3.8. Diagrama de classes	14
4. Conclusão	22
4.1. Lições aprendidas.....	22
4.2. Apreciação final.....	22
5. Bibliografia	23

1. Introdução

1.1. Contextualização

Este trabalho prático está inserido na unidade curricular de Linguagens de Programação II, do curso de Licenciatura de Engenharia de Sistemas Informáticos.

A realização deste trabalho prático consistiu na criação de um programa em linguagem C#, com o uso do Microsoft Visual Studio no sistema operativo Windows 10 home.

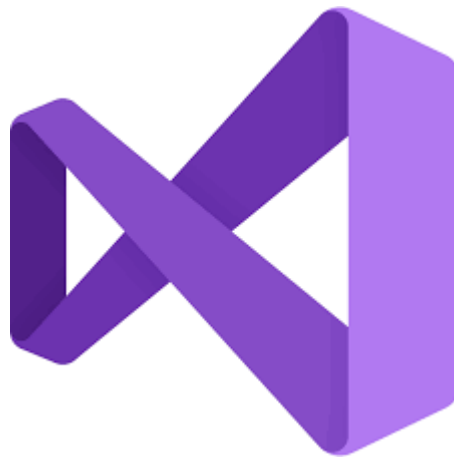


Figura 1 - Símbolo Microsoft Visual Studio

1.2. Motivação e objetivo

Cada vez mais as pessoas têm por hábito ser muito cuidadosas com os seus dados pessoais, como por exemplo quando os disponibilizam a empresas, ou a qualquer tipo de organização ou associação.

Quando uma empresa tem boa reputação por ser segura à nível de sistema informático, isso atrai clientes, porque eles próprios sentem-se seguros ao depositar lá os seus dados pessoais. O facto de atrair clientes é muito positivo para o negócio, daí esta empresa pretender contratar um programador para desenvolver um programa de auxílio às auditorias, para tornar a empresa mais segura e em conformidade com o RGPD.

1.3. Estrutura do Documento

O relatório deste trabalho prático encontra-se dividido em cinco capítulos:

1. **Introdução** – Neste capítulo, encontra-se um breve resumo sobre o que consiste este trabalho e quais os objetivos da realização deste mesmo;
2. **Estado da Arte** – No Estado da Arte está retratado o que é o RGPD e ainda algumas aplicações que existem para auxiliar as empresas a estarem de acordo com o RGPD;
3. **Implementação** – Na Implementação encontra-se uma descrição completa de todos os pormenores do trabalho, explicando cada fase, assim como o funcionamento do mesmo;
4. **Análise e Teste** – Na análise e testes está representado imagens sobre a execução do programa final, explicando detalhadamente cada passo e ainda uma análise final deste trabalho prático;
5. **Conclusão** - E por fim na conclusão fala sobre o que eu achei deste trabalho prático, quer a nível de dificuldades encontradas a meio do projeto e apreciação final sobre o que este trabalho melhorou em mim.

2. Estado da arte

O Regulamento Geral de Proteção de Dados foi aprovado em 15 de abril de 2016 no Parlamento Europeu e só foi implementado passados mais de dois anos em 25 de maio de 2018. Este regulamento consiste num conjunto de regras sobre a privacidade e proteção de dados pessoais de todos os cidadãos residentes na UE e a circulação deles para fora da UE e mesmo dentro da UE. Estas regras são aplicadas quer a pessoas, organizações ou empresas que operam com dados pessoais.

Com a implementação do RGPD, todas as empresas ativas na UE têm que respeitar as regras e os princípios definidos pelo RGPD, senão correm o risco de pagamento de coimas até 10 milhões de euros ou 2% do volume de negócio, no caso de infrações de menor gravidade e para casos mais graves poderá chegar aos 20 milhões de euros ou 4% do volume de negócios.

Existem algumas aplicações no mercado capazes de auxiliar auditorias, uma delas é a “Gestão de Auditorias” da WeMake, além de permitir a gestão online de todas as atividades ao planeamento e registo de auditorias (internas / externas), a aplicação produz automaticamente os documentos essenciais e armazena os registos associados.



Figura 2 - Regulamento Geral de Proteção de Dados

3. Implementação Fase 1

3.1. Descrição do problema

Neste trabalho prático, estamos perante o departamento informático de uma empresa de gestão e contabilidade, que pretende que um programador desenvolva um programa em C#. Esse programa terá como objetivo auxiliar os colaboradores responsáveis da empresa para quando forem realizadas auditorias internas, com diversos tipos de funcionalidades.

3.2. Solução

Para iniciar o trabalho, eu comecei por organizar este projeto em cinco ficheiros diferentes, onde cada um corresponde a uma parte do programa, tais como as auditorias, colaboradores, vulnerabilidades e equipamentos.

3.3. Program.cs

O ficheiro “Program.cs”, vai ser o ficheiro principal deste programa, onde todas as classes vão girar a volta dele.

Nesta fase do trabalho prático, ainda não temos nada desenvolvido neste ficheiro, pois terá como função passar cá para fora a informação, onde essa parte vai ser desenvolvida na Fase 2 deste trabalho prático

3.4. Auditoria.cs

Este primeiro ficheiro criado “Auditoria.cs”, contém dentro dele tudo que diz respeito às auditorias.

Primeiro comecei por introduzir as variáveis que caracterizam as auditorias:

- **int:**
 - **codigo** → representa o código da auditoria;
 - **duracao** → corresponde a duração da auditoria;
 - **codColaborador** → corresponde ao código do colaborador que realizou a auditoria;

- **DateTime** :
 - ***data*** → data que se realizou a auditoria;
- **Vulnerabilidade[]** :
 - ***vulns*** → conjunto de todas as vulnerabilidades presentes numa auditoria;
- **static int** : variáveis criadas para serem utilizadas dentro da classe auditoria
 - ***countCodigo*** → contador utilizado para atribuir os códigos automaticamente as auditorias;
 - ***countVulnerabilidade*** → contador utilizado para contar as vulnerabilidades de uma auditoria;

Depois passei para a parte da criação dos construtores. Onde criei um método static, para ser executado pelo programa, onde vai inicializar as variáveis static int com 0 e criei dois métodos public, para construir a auditoria, onde um é construção por defeito e outro é construção com valores.

A seguir aos construtores, avancei para a criação das propriedades com o uso de get's e set's, onde é possível alterar os valores das variáveis da classe Auditoria, ou então apenas verificar qual o valor da mesma.

Nas propriedades, criei métodos que permitem alterar e verificar para duracao, codColaborador e data. Depois criei para codigo, um que apenas permite verificar o valor da variável da auditoria.

Por último, desenvolvi quatro funções:

- ***InserirVulnAuditoria()*** → tem como funcionalidade passar uma vulnerabilidade para dentro de uma auditoria;
- ***NumeroAuditorias()*** → devolve o número de auditorias presentes no programa;
- ***AuditoraiMaisVulnerabilidades()*** → encontra e devolve a auditoria com mais vulnerabilidades;
- ***AuditoraiMenosVulnerabilidades()*** → encontra e devolve a auditoria com menos vulnerabilidades;

3.5. Colaborador.cs

Neste ficheiro “Colaborador.cs”, encontra-se tudo que envolve os colaboradores e tudo que eles têm de envolvido à volta deles.

Primeiro comecei por criar um enum, para declarar o estado de um colaborador, de seu nome State e pode tomar dois valores:

- *ativo* → onde se encontra ativo para realizar uma auditoria;
- *inativo* → onde não se encontra disponível para realizar uma auditoria;

Depois introduzi as variáveis que identificam os colaboradores:

- **int:**
 - **codigo** → representa o código do colaborador;
- **string:**
 - **nome** → representa o nome do colaborador;
- **State:**
 - **estado** → corresponde ao estado de atividade do colaborador;
- **Auditoria[] :**
 - **audColaborador** → conjunto de todas as auditorias presentes num colaborador;
- **static int :** variáveis criadas para serem utilizadas dentro da classe Colaborador.
 - **countCodigo** → contador utilizado para atribuir os códigos automaticamente aos colaboradores;
 - **countAuditoria** → contador utilizado para contar as auditorias num colaborador;

A seguir avancei para a parte do desenvolvimento dos construtores. Onde criei um método static, para ser executado pelo programa, onde vai inicializar as variáveis static int com 0 e criei dois métodos public, para construir a Vulnerabilidade, onde um é construção por defeito e outro é construção com valores.

Depois da criação dos construtores, desenvolvi as propriedades com o uso de get's e set's, onde é possível alterar os valores das variáveis da classe Colaborador, ou então apenas verificar qual o valor da mesma.

Dentro das propriedades, criei métodos que permitem alterar e verificar os valores das variáveis para estado e nome. Depois criei para código, um que apenas permite verificar o valor da variável do colaborador.

Para terminar, desenvolvi duas funções:

- **InserirAudColaborador()** → permite adicionar uma auditoria ao registo de auditorias dentro de um colaborador;
- **RemoverColaborador()** → altera o estado de atividade de um colaborador para inativo;

3.6. Vulnerabilidade.cs

Neste terceiro ficheiro “Vulnerabilidade.cs”, contem tudo que envolve as vulnerabilidades.

Antes de começar a declarar as variáveis, criei dois enum, para declarar o estado e o nível de uma vulnerabilidade, são essas:

- **Level**
 - *baixo*;
 - *moderado*;
 - *elevado*;
- **State**
 - *sim* → representa que a vulnerabilidade já se encontra resolvida;
 - *nao* → representa que a vulnerabilidade não se encontra resolvida;

A seguir passei para a declaração das variáveis da classe Vulnerabilidade:

- **int:**
 - ***codigo*** → representa o código da vulnerabilidade;
- **string:**
 - ***descricao*** → corresponde a descrição da vulnerabilidade;
- **State:**
 - ***estado*** → corresponde ao estado de resolução da vulnerabilidade;
- **Equipamento[] :**
 - ***eqVuln*** → conjunto de equipamentos presentes numa vulnerabilidade;
- **int[]:**
 - ***codAuditoria*** → conjunto de códigos de auditoria presentes numa vulnerabilidade;

- **static int** : variáveis criadas para serem utilizadas dentro da classe Vulnerabilidade.
 - **countCodigo** → contador utilizado para atribuir os códigos automaticamente aos colaboradores;
 - **countAuditoria** → contador utilizado para contar as auditorias num colaborador;
 - **countEquipamento** → contador utilizado para contar as auditorias num colaborador;

Depois avancei para a criação dos construtores. Onde criei um método static, para ser executado pelo programa, onde vai inicializar as variáveis static int com 0 e criei dois métodos public, para construir a vulnerabilidade, onde um é construção por defeito e outro é construção com valores, onde introduz uma string descrição, um Level do nível e um State do estado.

Depois dos construtores, passei para a criação das propriedades com o uso de get's e set's, onde é possível alterar os valores das variáveis da classe Vulnerabilidade, ou então apenas verificar qual o valor da mesma.

Nas próprias propriedades, criei métodos que permitem alterar e verificar os valores das variáveis para estado, nivel, descricao e codAuditoria. Depois criei para codigo, um que apenas permite verificar o valor da variável da vulnerabilidade.

Para o fim, desenvolvi três funções:

- **InserirAudVulnerabilidade()** → permite adicionar uma auditoria ao registo de auditorias dentro de uma vulnerabilidade;
- **RemoverVulnerabilidade()** → altera o estado de realização de uma vulnerabilidade para sim;
- **InserirEquipVulnerabilidades ()** →adiciona um equipamento a variável eqVuln de uma vulnerabilidade;

3.7. Equipamento.cs

Neste último ficheiro "Equipamento.cs", vai conter tudo que diz respeito aos equipamentos.

Primeiro comecei por introduzir as variáveis que caracterizam os equipamentos:

- **int**:
 - **codigo** → representa o código do equipamento;

- **string:**
 - ***tipo*** → representa o tipo de equipamento;
 - ***marca*** → corresponde a marca do modelo;
 - ***modelo*** → representa o modelo do equipamento;
- **int[]:**
 - ***codVuln*** → conjunto de códigos de vulnerabilidade presentes num equipamento;
- **DateTime :**
 - ***dataAquisicao*** → data em que se comprou o equipamento;
- **static int :** variáveis criadas para serem utilizadas dentro da classe Vulnerabilidade.
 - ***countCodigo*** → contador utilizado para atribuir os códigos automaticamente aos equipamentos;
 - ***countVuln*** → contador utilizado para contar as vulnerabilidades presentes num equipamento;

A seguir passei para a parte do desenvolvimento dos construtores. Onde criei um método static, para ser executado pelo programa, onde vai inicializar as variáveis static int com 0 e criei dois métodos public, para construir o Equipamento, onde um é construção por defeito e outro é construção com valores, onde vai ser introduzido quatro string's tipo, marca, modelo e data de aquisição que depois vai ser convertido para DateTime, através do .Parse().

Depois da criação dos construtores, desenvolvi as propriedades com o uso de get's e set's, onde é possível alterar os valores das variáveis da classe Equipamento, ou então apenas verificar qual o valor da mesma.

Dentro das propriedades, criei métodos que permitem alterar e verificar os valores das variáveis para tipo, marca, modelo e dataAquisicao. Depois criei para codigo, um que apenas permite verificar o valor da variável do equipamento.

Para terminar, desenvolvi uma função:

- **InserirVulnEquipamento ()** → permite adicionar uma vulnerabilidade ao registo de vulnerabilidades dentro de um equipamento;

3.8. Diagrama de classes

Na imagem seguinte, encontra-se representado o digrama desta fase inicial do trabalho pratico, onde contém todas variáveis, propriedades, funções e enum's de cada classe e ainda as ligações entre eles:

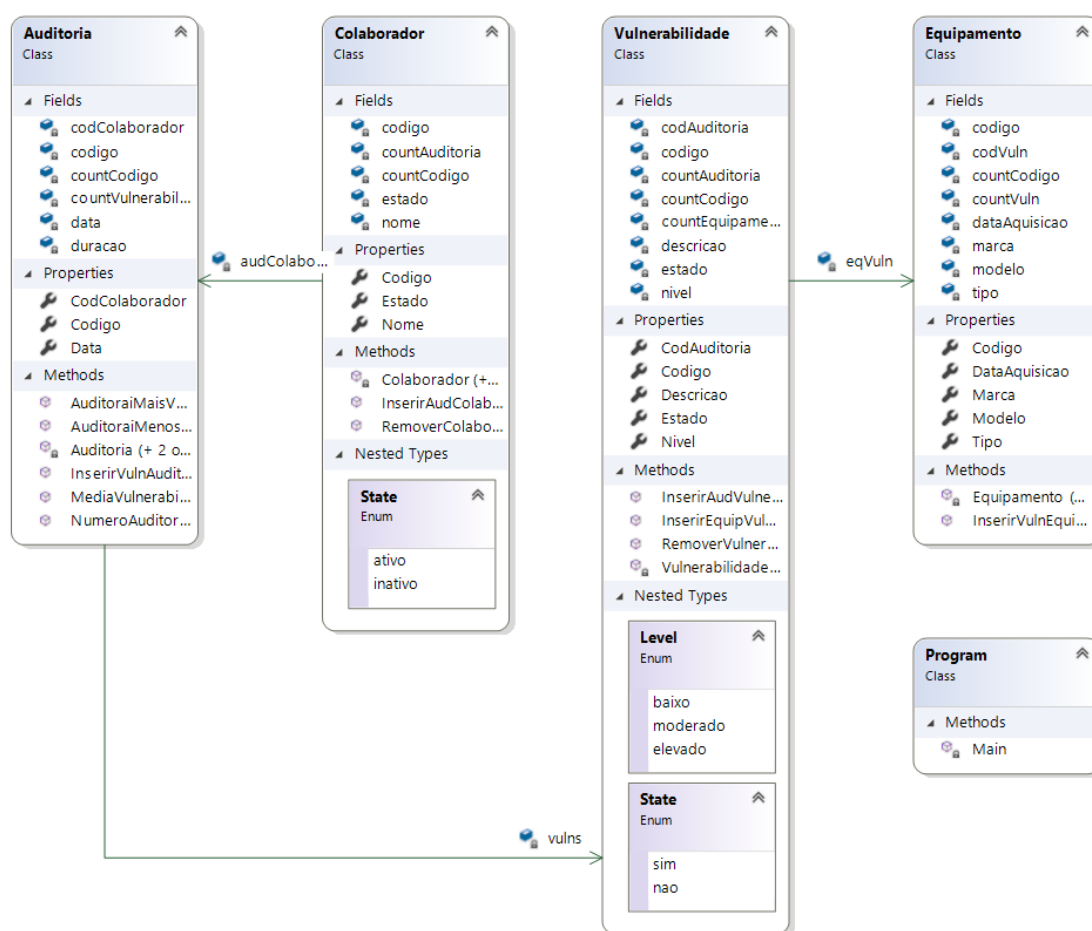


Figura 3 - diagrama de classes

4. Implementação Fase 2

Esta fase 2 do trabalho prático consistiu em melhorar a fase 1 e aplicar nova matéria que não tinha sido dada até a data.

4.1. NTier

Nesta nova fase do trabalho prático, implementei a arquitetura **Ntier**, no qual tenho cinco projetos na mesma solução, respeitando as normas da arquitetura:

- **Tp1Lp2** – Presentation Layer;
- **ObjetosAuditoria** - Business Objects;
- **BDAuditorias** – Data Access Objects;
- **RegrasNegocio** – Business Rules;
- **Excepcoes**;

Para melhorar a qualidade do código substitui todos os arrays por listas. Como por exemplo:

```
Vulnerabilidades[] vulns; => List<Vulnerabilidade> vulns;
```

4.2. Exceções

Com a introdução desta matéria relativa as exceções, conseguimos lidar com algumas situações que surgem quando o programa esta a decorrer, através do try{}catch{}. Onde permite captar falhas que possam surgir durante a execução do programa.

Neste trabalho para além das exceções já predefinidas pelo visual studio, desenvolvi algumas para tentar captar alguns erros específicos.

```
public class AuditoriaInexistenteException : ApplicationException -caso não exista  
a auditoria que o programa esta a tentar manipular;
```

```
public class VulnerabilidadeInexistenteException : ApplicationException -caso não  
exista a vulnerabilidade que o programa esta a tentar manipular;
```


`public class ColaboradorInexistenteException : ApplicationException` -caso não exista o colaborador que o programa esta a tentar manipular;

`public class EquipamentoInexistenteException : ApplicationException` -caso não exista o equipamento que o programa esta a tentar manipular;

`public class InsercaoException : ApplicationException` - se o programa não conseguir fazer a inserção de algum tópico;

4.3. Diagramas de Classe

Diagrama Objetos Auditoria

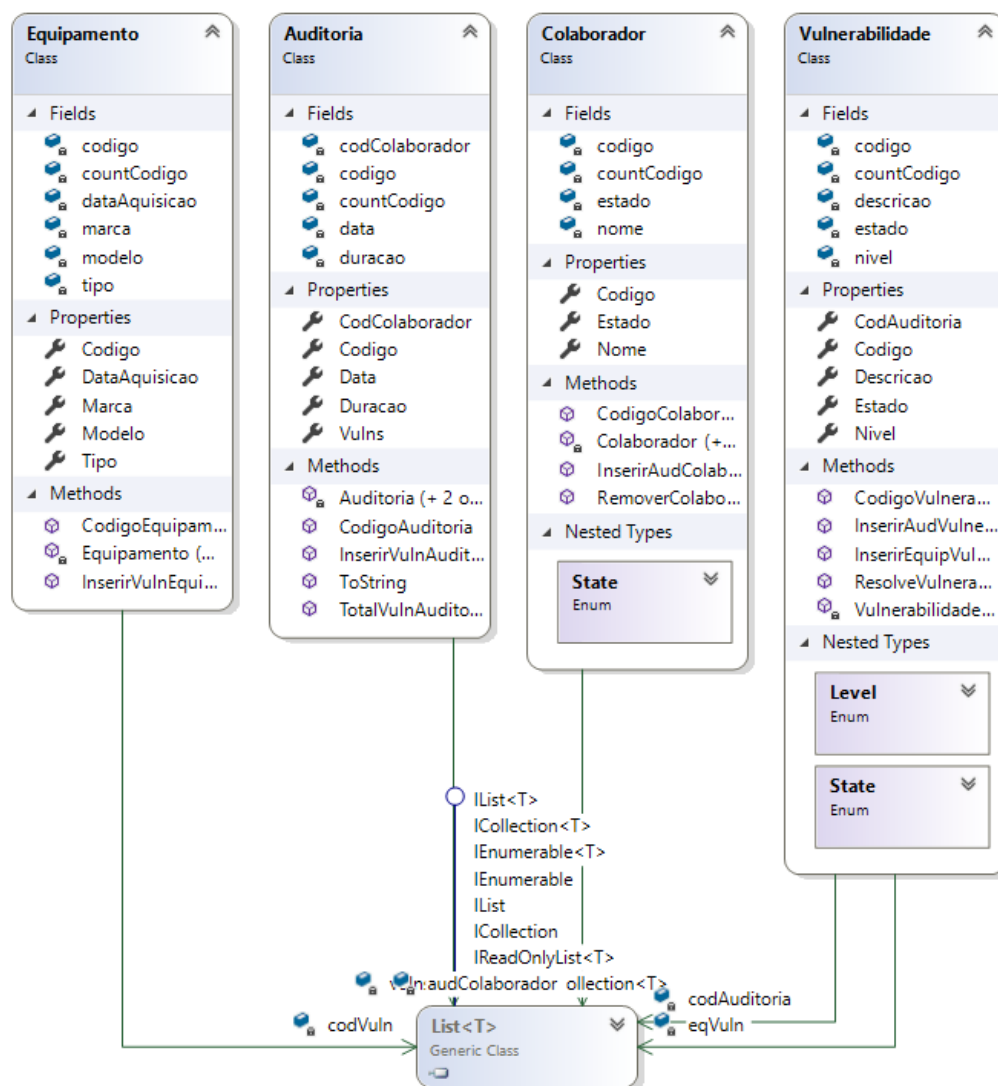


Figura 4 - Diagrama Objetos Auditoria

Diagrama BDAuditorias

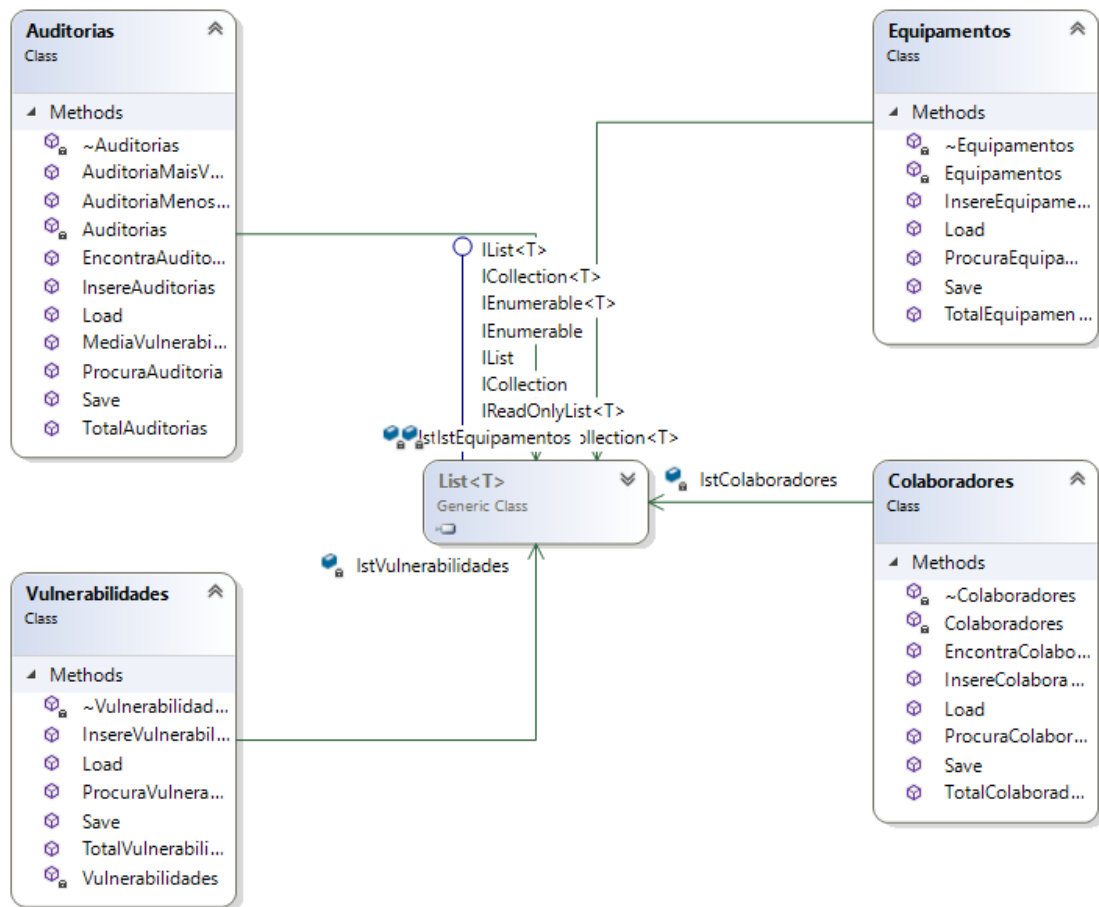


Figura 5 -Diagrama BDAuditorias

Diagrama Exceções

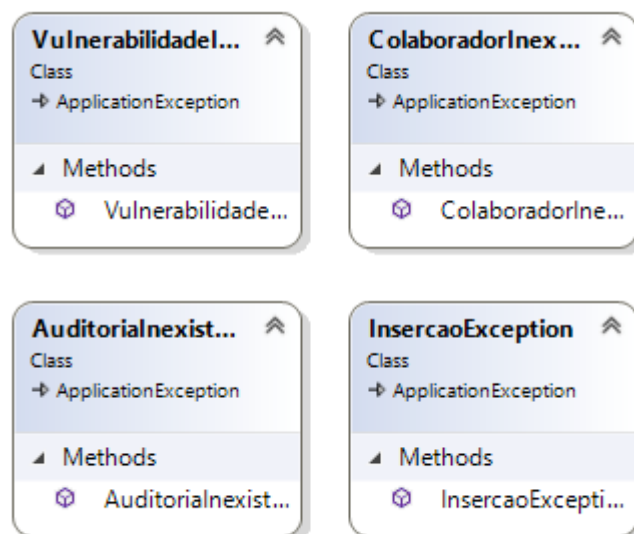


Figura 6 - Diagrama Exceções

5. Implementação Fase 3

Nesta última fase do trabalho prático, houve ligeiras correções do código e finalização de alguns tópicos que faltavam realizar no trabalho.

5.1. Camada de Apresentação Tp1Lp2

Neste projeto, foi desenvolvido a “presentation layer” do programa, onde nesta fase foi finalizado tudo o que faltava desta no “Program.cs”.

Adicionei na classe Main três colaboradores, três auditorias, três vulnerabilidades e dois equipamentos para testar o funcionamento do programa. De seguida chamei algumas funções presentes nas regras de negócio para adicionar cada um a sua respetiva lista e fazer a respetiva ligação entre eles:

- `RN.InsereAuditoria(a);`
- `RN.InsereColaborador(jose);`
- `RN.InsereVulnerabilidade(va);`
- `RN.InsereEquipamento(asusZenbook);`
- `RN.InsereVulnerabilidadeAuditoria(a, va);`
- `RN.InsereColaboradorAuditoria(a, jose);`
- `RN.InsereVulnerabilidadeEquipamento(va, asusZenbook);`

A seguir chamei um método desenvolvido nesta última fase no projeto regras de negocio, onde apresenta uma serie de dados sobre o programa onde vai conter tudo sobre os resultados obtidos:

- `RN.ApresentarDados();`

Por último, coloquei dois métodos a serem testados, para resolver vulnerabilidade e alterar o estado de atividade do colaborador e de seguida voltei a apresentar os dados sobre o programa, para analisar os resultados dos métodos:

- `RN.ResolveVulnerabilidade(va);`
- `RN.ResolveColaborador(luis);`

5.2. Ficheiros Binários

5.3. Correção de erros

Algumas correções relativas às outras fases:

Substituição de listas de objetos por listas de inteiros

Até esta fase, em alguns projetos, tinha dentro de um objeto listas de outros objetos. Depois de um estudo sobre o caso, alterei as listas de objetos para listas de inteiros que contem o código da respetiva auditoria, vulnerabilidade, equipamento ou colaborador.

```
private List<Vulnerabilidade> codVulns; => private List<int> codVulns;  
  
private List<Equipamento> codEqVuln; => private List<int> codEqVuln;  
  
private List<Auditoria> codAuditoria; => private List<int> codAuditoria;  
  
private List<Auditoria>codAudColaborador;=private List<int>codAudColaborador;
```

Alteração nos Enums

No início tinha os enums em minúscula e efetuei nesta fase a respetiva correção, onde coloquei todos em maiúscula:

ativo => ATIVO

inativo => INATIVO

Substituições nas Exceções

Quando criei alguns métodos com exceções, coloquei a imprimir o erro através do método "Console.WriteLine()", nesta fase alterei para "throw new «exception»()".

```
catch (Exception e)  
{  
    throw new Exception("ERRO! " + e.Message);  
}
```

5.4. Diagrama de Classes

Diagrama ObjetosAuditoria

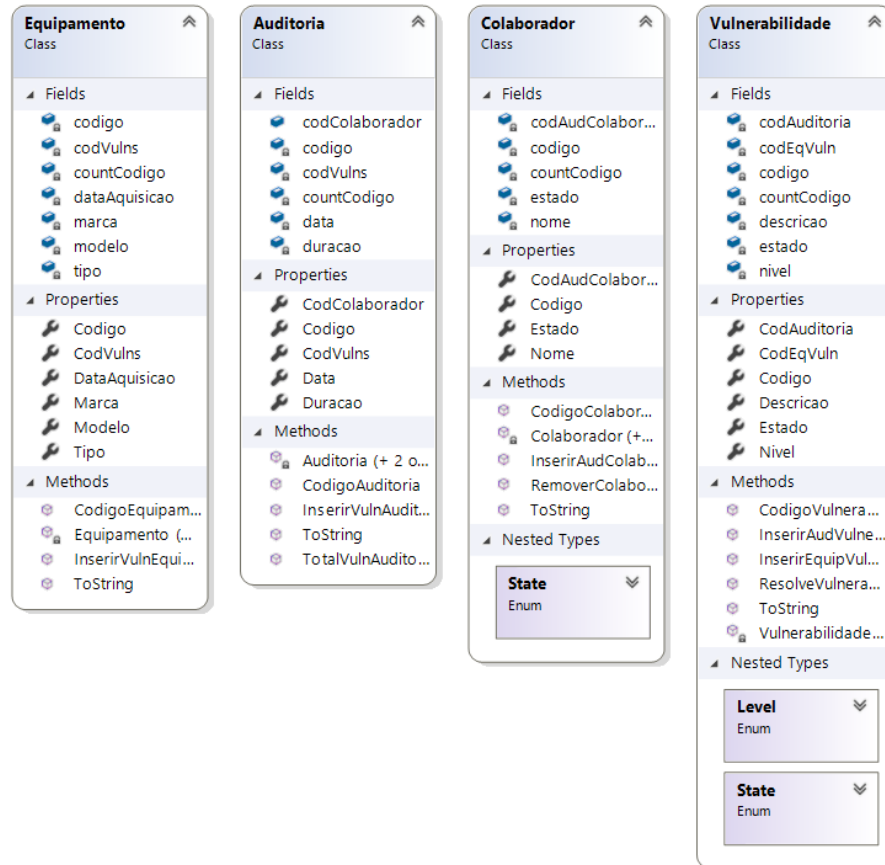


Figura 7 - Diagrama ObjetosAuditoria

Diagrama BDAuditorias

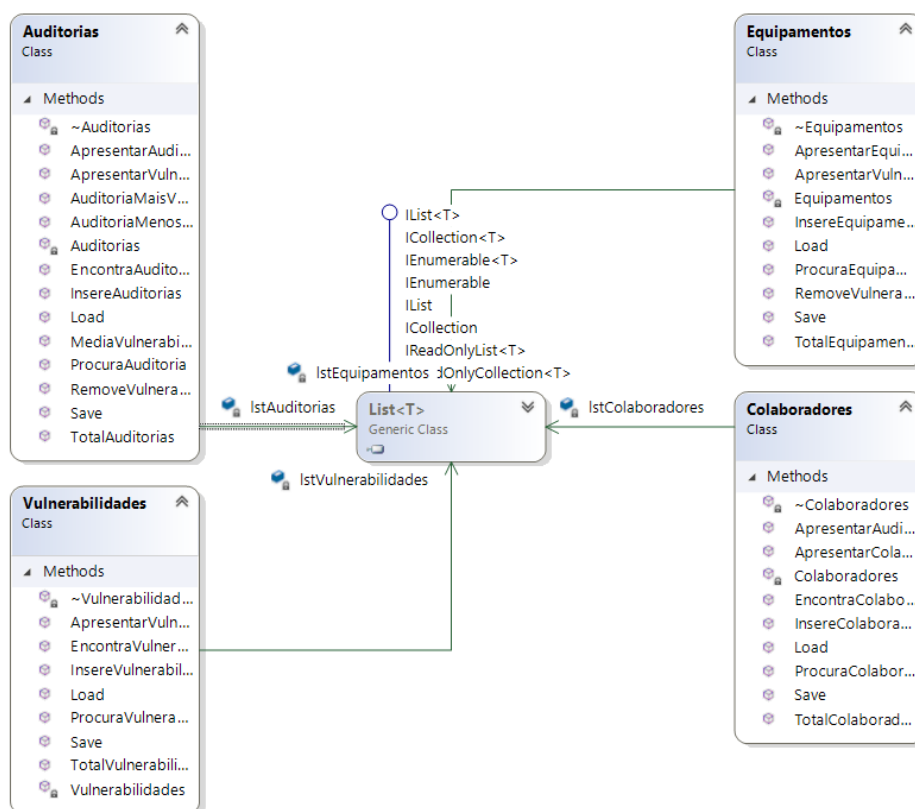


Figura 8 - Diagrama BDAuditorias

Diagrama Exceções

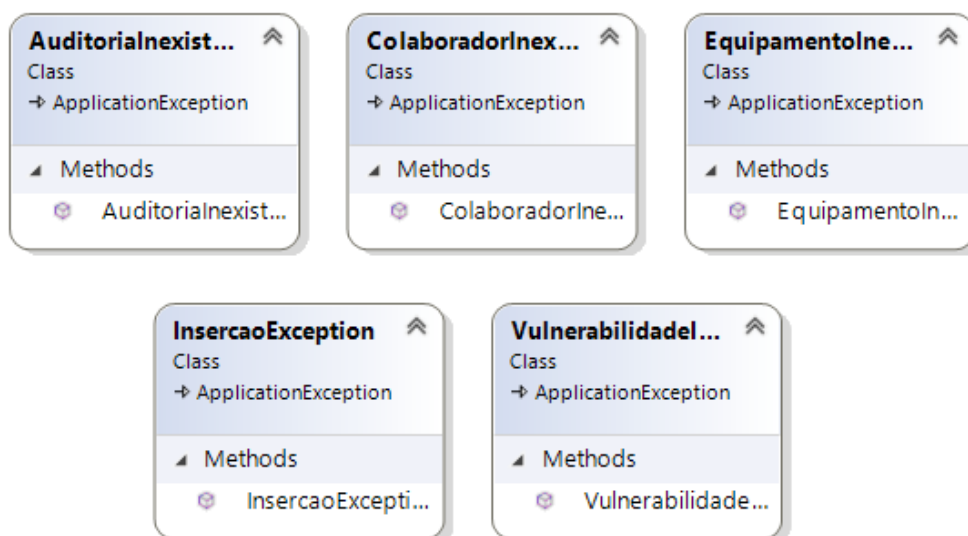


Figura 9- Diagrama Exceções

6. Conclusão

6.1. Lições aprendidas

Este trabalho prático foi bastante difícil e complicado de fazer, visto que foi o primeiro realizado com linguagem C#.

Para a realização deste programa, tive de investigar matérias relacionadas com a linguagem C# que não tinha ainda dado nas aulas, pois necessitei delas e ainda estudei matéria que não usei no trabalho, mas fiquei a conhecer e saber operar com elas. O que na minha opinião, é muito benéfico, pois ganhei mais conhecimento na linguagem C# do que aquele que tinha.

6.2. Apreciação final

O trabalho prático desenvolvido nesta fase final do semestre tem um grande peso da avaliação final deste trabalho prático, daí eu ter me aplicado bastante, onde despendi muito tempo, empenho e dedicação.

Como este trabalho prático é o primeiro desta unidade curricular, acho que superei as minhas expectativas, tendo feito todo o trabalho da maneira que eu tinha pretendido, sem ter de fazer rodeios para obter o resultado desejado.

7. Bibliografia

“Regulamento Geral sobre a Proteção de Dados”, 07/11/2019 ,Wikipedia:

https://pt.wikipedia.org/wiki/Regulamento_Geral_sobre_a_Prote%C3%A7%C3%A3o_de_Dados