INSTITUTO POLITÉCNICO
DO CÁVADO E DO AVE
ESCOLA SUPERIOR DE TECNOLOGIA

RELATÓRIO DE TRABALHO PRÁTICO

Sistema de Registo de Auditorias

CARLOS SANTOS

ALUNO Nº 19432

RÚBEN SILVA

ALUNO Nº 19433

Trabalho realizado sob a orientação de:
Luís Ferreira

Linguagens de Programação II

Licenciatura em Engenharia de Sistemas Informáticos

Barcelos, abril de 2020

Índice

Lista de Figuras

1 Implementação

1.1 Descrição do Problema

Necessidade de criar um programa que auxilie colaboradores de uma empresa na criação de auditorias internas. Sendo que este deve oferecer uma vasta quantidade de funcionalidades para o utilizador, tais como, a inserção e edição de colaboradores e vulnerabilidades, a remoção de colaboradores, a edição das vulnerabilidades de uma auditoria, entre outras. Para mostrar resultados e dados, o programa deve também dar a possibilidade de apresentar diferentes listas, como uma lista onde as auditorias estão ordenadas pela quantidade de vulnerabilidades ou então numa auditoria, estas vulnerabilidades estão agrupadas por nível de impacto. Também devem ser apresentados os detalhes de um colaborador mostrando as auditorias realizadas por este, e também mostrar as vulnerabilidades identificadas num equipamento.

Por fim, deve também ser apresentado uma tabela geral onde devem ser comparadas todas as auditorias mostrando a que tem mais e a que tem menos vulnerabilidades, a quantidade de auditorias e a média de vulnerabilidade em todas as auditorias registadas.

1.2 Solução

Para solucionar o problema, nesta primeira fase do programa, foram criadas as classes necessárias para resolver este problema, sendo elas, auditoria, equipamento, vulnerabilidade e colaborador que tem como classe base pessoa.

Mais tarde serão criadas as funções necessárias para solucionar o problema.

2 Estado de Arte

O Regulamento Geral sobre a Proteção de Dados entrou em vigor em 25 de maio de 2018 e é um regulamento do direito europeu sobre privacidade e proteção de dados pessoais, aplicável a todos os indivíduos na União Europeia, regulamenta também os dados pessoais exportados para fora da União Europeia.
Este regulamento obriga a informar acerca da base legal para o tratamento de dados, prazo de conservação dos mesmos e a sua transferência.
O regulamento obriga a controlar as circunstâncias em que foi obtido o consentimento dos titulares quando isso for base legal do tratamento dos dados pessoais. Existem um conjunto de exigências para obtenção desse consentimento e o seu não cumprimento obriga à obtenção de um novo consentimento.
O regulamento obriga a um grande controlo do risco associado ao possível roubo de informação. Este controlo de risco deverá passar a ser garantido por medidas de segurança efetivas que garantam a confidencialidade, a integridade dos dados e que previnam a destruição, ou a divulgação/acesso não autorizado de dados.
Existem DPO's – Data Protection Officer – (Delegado de Proteção de Dados) que são um canal de comunicação entre uma determinada empresa e uma agência de Proteção de Dados e os titulares dos dados pessoais e têm como função auxiliar a empresa no processo de adaptação e estruturação de um programa com foco na proteção de dados, que tem de se manter a par de todas as obrigações do RGPD. Estes podem usar várias aplicações que os ajudam nesta tarefa, como por exemplo, a GDRP App.

3 Classes

Abaixo estão representadas as classes implementadas no programa.

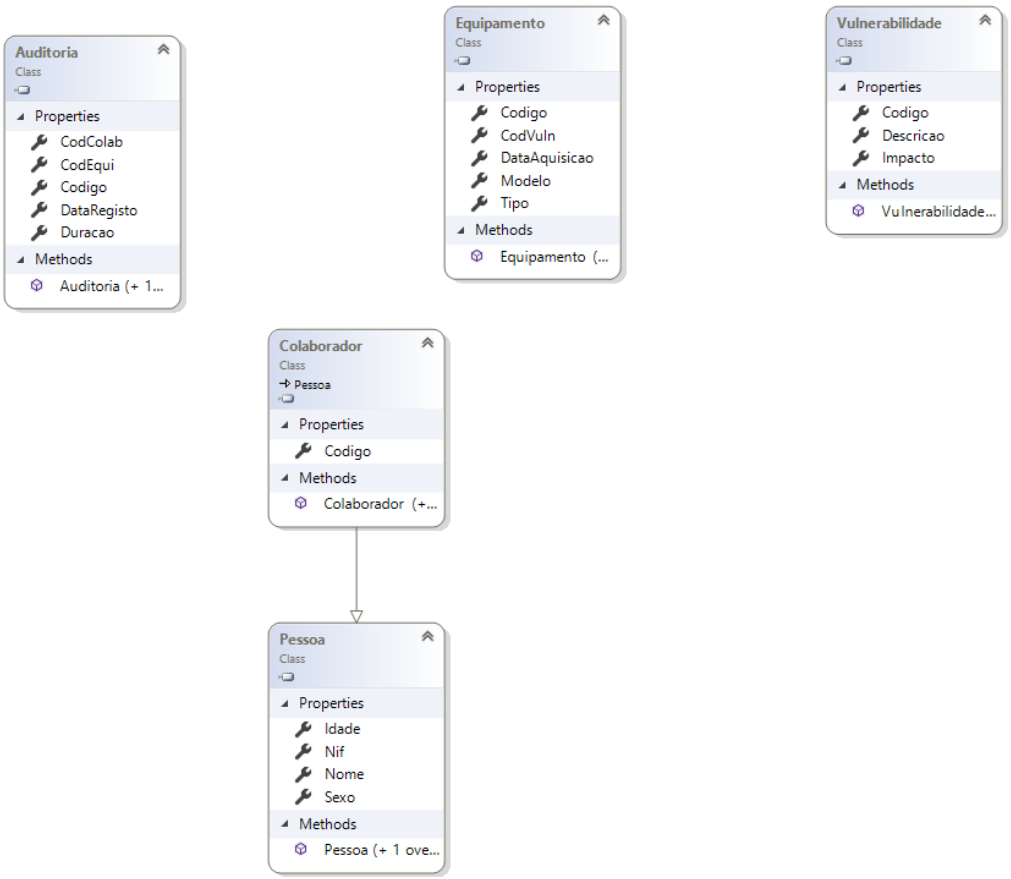

Figura 1: Diagrama de Classes

3.1 Estado

Estado é a região onde serão criadas todas as variáveis necessárias para cada classe.

3.1.1 Auditoria

int *codigo* -> Código da Auditoria;

int *duracao* -> Duração da Auditoria;

DateTime dataRegisto -> Data em que a Auditoria ocorreu;

int *codColab* -> Código do colaborador responsável pela auditoria;

int *codEqui* -> Código do Equipamento ao qual foi realizada a auditoria.

3.1.2 Colaborador

int *codigo* -> Código referente ao colaborador.

3.1.3 Equipamento

int *codigo* -> Código referente ao Equipamento;

string *tipo*-> Tipo de Equipamento;

string *modelo* -> Modelo do Equipamento;

DateTime *dataAquisicao* -> Data de Aquisição do Equipamento;

int *codVuln* -> Codigo da Vulnerabilidade detetada no Equipamento;

3.1.4 Pessoa

string *sexo* -> Sexo referente à pessoa;

string *nome* -> Nome pessoal;

int *idade* -> Idade;

int *nif* -> Número de identificação fiscal;

3.1.5 Vulnerabilidade

int *codigo* -> Código da Vulnerabilidade;

string *descricao* -> Descrição da Vulnerabilidade em causa;

NivelImpacto *impacto* -> Nível de impacto da vulnerabilidade.

3.2Métodos

A região “Métodos” está relacionada com a manuseamento das variáveis.

3.2.1 Auditoria

São criados 2 construtores, o primeiro está responsável pela inicialização por defeito das variáveis (int *codigo*, int *duracao*, DateTime *dataRegisto*, int *codColab*, int *codEqui*), o segundo recebe valores externos (int *codigo*, int *duracao*, DateTime *dataRegisto*, int *codColab*, int *codEqui*).

É também criada a região propriedades que servirá para manipular os atributos das auditorias.

3.2.2 Colaborador

Inicialmente é criado um construtor para inicializar a variável (int *codigo*) por defeito. Posteriormente outro construtor irá receber valores externos (int *codigo*, string *nome*, string *sexo*, int *idade*, int *nif*): base (*nome*, *sexo*, *idade*, *nif*).

No final, o atribuído da variável (int *codigo*) é alterado.

3.2.3 Equipamento

São criados 2 construtores, o primeiro está responsável pela inicialização por defeito das variáveis (int *codigo*, string *tipo*, DateTime *dataRegisto*, string *modelo*, int *codVuln*), o segundo recebe valores externos (int *codigo*, string *tipo*, string *modelo*, DateTime *dataAquisicao*).

É também criada a região propriedades que servirá para manipular os atributos das auditorias.

3.2.4 Pessoa

É criado um construtor para inicializar a variável (int *nif,* int *idade*, string *nome*, string *sexo*) por defeito. Posteriormente outro construtor irá receber valores externos (string *nome*, string *sexo*, int *idade*, int *nif*).

O atributo das variáveis (string *sexo*, string *nome*, int *nif*, *int Idade*) também sofrem manipulação.

3.2.5 Vulnerabilidade

São criados 2 construtores, o primeiro está responsável pela inicialização por defeito das variáveis (int *codigo*, string *descricao*, NivelImpacto *impacto*), o segundo recebe valores externos (int *codigo*, int *duracao*, DateTime *dataRegisto*, int *codColab*, int *codEqui*)

É também criada a região propriedades que servirá para manipular os atributos das auditorias.

4 Funções

4.1 Atributos

Atributos é a região onde serão atribuídos valores a todas a variáveis necessárias para as funções.

4.1.1 Auditoria

const int *MAX* = 100;

static Auditoria [] *aud*;

static int *totalAud*=0;

4.1.2 Colaborador

const int *MAX* = 100;

static Colaborador [] *col*;

static int *totalCol*=0;

4.1.3 Equipamento

const int *MAX* = 100;

static Equipamento [] *equ*;

static int *totalEqu* = 0;

4.1.4 Pessoa

const int *MAX* = 100;

static Pessoa [] *pes*;

static int *totalPes* = 0;

4.1.5 Vulnerabilidade

const int *MAX* = 100;

static Vulnerabilidade [] *vul*;

static int *totalVul*;

4.2 Métodos

A região "Métodos" está relacionada com a manuseamento das variáveis.

4.2.1 Auditoria

Inicialmente é criado o "default Constructor" (Auditoria [] *aud*). Depois são implementadas 3 funções:

public static int RegistaAuditoria (Auditoria *a*) -> Responsável pelo registo de uma auditoria.

aud[totalAud++] = a; -> adiciona uma nova auditoria ao array de auditorias.

Esta função começa por verificar se a Auditoria tem espaço no array reservado para a guardar, se este existir, irá passar pela função "ExisteAuditoria" para verificar se essa auditoria está presente no array, se estiver RETURN -1, se for nova, Return 1;

private static bool ExisteAuditoria(int *cod*) -> Responsável pela verificação da Auditoria no Array.

Esta função irá percorrer todas as auditorias, se encontrar uma auditoria repetida, RETURN true, se não existir, RETURN false... Esta função é implementada na função "ResgistaAuditoria".

private static bool MostraAuditoria (int *cod*) -> Responsável pela apresentação da Auditoria na Consola.

Esta função mostra na consola as informações relacionadas com a auditoria.

4.2.2 Colaborador

Inicialmente é criado o "default Constructor" (Colaborador [] *col*). Depois são implementadas 2 funções:

public static int RegistaColaborador (Colaborador c) -> Responsável pelo registo de um colaborador

col[totalCol++] = c; -> adiciona um novo colaborador ao array de colaboradores.

Esta função começa por verificar se o Colaborador tem espaço array reservado para guardar-los, se este existir, irá passar pela função "ExisteColaborador" para verificar se essa auditoria está presente no array, se estiver RETURN -1, se for nova, Return 1;

private static bool ExisteColaborador(int cod) -> Responsável pela verificação do Colaborador no Array.

Esta função irá percorrer todos os colaboradores, se encontrar um colaborador repetido, RETURN true, se não existir, RETURN false... Esta função é implementada na função "ResgistaColaborador".

4.2.3 Equipamento

Inicialmente é criado o "default Constructor" (Equipamento [] *equ*). Depois são implementadas 2 funções:

public static int RegistaEquipamento(Equipamento e) -> Responsável pelo registo de um Equipamento.

equ[totalEqu++] = e; -> adiciona um novo equipamento ao array de equipamentos.

Esta função começa por verificar se o Equipamento tem espaço array reservado para guardar-los, se este existir, irá passar pela função "ExisteEquipamento" para verificar se esse Equipamento está presente no array, se estiver RETURN -1, se for nova, Return 1;

private static bool ExisteEquipamento(int cod) -> Responsável pela verificação do Equipamento no Array.

Esta função irá percorrer todas os Equipamentos, se encontrar um Equipamento repetido, RETURN true, se não existir, RETURN false... Esta função é implementada na função "ResgistaEquipamento".

4.2.4 Pessoa

Inicialmente é criado o "default Constructor" (Pessoa[] *pes*). Depois são implementadas 2 funções:

private static bool ExistePessoa(int nif) -> Responsável pelo registo de uma Pessoa.

pes[totalPes++] = c; -> adiciona uma nova pessoa ao array de pessoas.

Esta função começa por verificar se a Pessoa tem espaço array reservado para as guardar, se este existir, irá passar pela função "ExistePessoa" para verificar se essa Pessoa está presente no array, se estiver RETURN -1, se for nova, Return 1;

public static int RegistaPessoa(Pessoa p) -> Responsável pela verificação da Pessoa no Array.

Esta função irá percorrer todas as Pessoas, se encontrar uma Pessoa repetida, RETURN true, se não existir, RETURN false... Esta função é implementada na função "ResgistaPessoa".

4.2.5 Vulnerabilidade

Inicialmente é criado o "default Constructor" (Vulnerabilidade [] *vul*). Depois são implementadas 2 funções:

public static int RegistaVulnerabilidade(Vulnerabilidade v) -> Responsável pelo registo de uma Vulnerabiliadde.

vul[totalVul++] = v; -> adiciona uma nova vulnerabilidade ao array de vulnerabilidaes.

Esta função começa por verificar se a Vulnerablidade tem espaço array reservado para as guardar, se este existir, irá passar pela função "ExisteVulnerablidade" para verificar se essa Vulnerabilidade está presente no array, se estiver RETURN -1, se for nova, Return 1;

private static bool ExisteVulnerabilidade(int cod) -> Responsável pela verificação da Vulnerabilidade no Array.

Esta função irá percorrer todas as Vulnerabilidades, se encontrar uma Vulnerabilidade repetida, RETURN true, se não existir, RETURN false... Esta função é implementada na função "ResgistaVulnerabilidade".

5 Conclusão

O programa nesta fase inicial apresenta todos os requisitos e encontra-se visualmente agradável e de forma organizada. O código tem os comentários necessários para que qualquer pessoa consiga facilmente perceber do que cada parte se trata.

Com esta parte do trabalho prático, as nossas capacidades com a linguagem C# foram remodeladas e melhoradas principalmente no que toca a classes e ao paradigma POO.

O programa encontra-se alojado no seguinte repositório GitHub: https://github.com/carlosantos2103/19432_19433_LP2

Bibliografia

https://mydataprivacy.eu/o-regulamento/

https://eur-lex.europa.eu/legal-content/PT/TXT/HTML/?uri=OJ:L:2016:119:FULL

https://gdprapp.com/