**UNIVERSIDADE PAULISTA**
**ANALISE E DESENVOLVIMENTO DE SISTEMAS (ADS)**

**BRUNO DE OLIVEIRA FRANCISCO**
**HUALTER DE SOUZA FERREIRA DO NASCIMENTO**
**SAMUEL DOUGLAS CARNEIRO**

# OUVINIP

**Projeto Integrado Multidisciplinar IV**

**SOROCABA**
**2018**

**BRUNO DE OLIVEIRA FRANCISCO**
**HUALTER DE SOUZA FERREIRA DO NASCIMENTO**
**SAMUEL DOUGLAS CARNEIRO**

# OUVINIP

## Projeto Integrado Multidisciplinar IV

Trabalho de conclusão de semestre na forma de Projeto integrado Multidisciplinar (PIM) apresentado à Universidade Paulista – UNIP.

Orientador – Todos os professores do semestre.

**SOROCABA**
**2018**

**BRUNO DE OLIVEIRA FRANCISCO**
**HUALTER DE SOUZA FERREIRA DO NASCIMENTO**
**SAMUEL DOUGLAS CARNEIRO**

# OUVINIP

## Projeto Integrado Multidisciplinar IV

Trabalho de conclusão de semestre na forma de Projeto integrado Multidisciplinar (PIM) apresentado à Universidade Paulista – UNIP.

Orientador – Todos os professores do semestre.

**Nota:**

_____/___/___
**Prof. Reverdan Sparinger**
**UNIVERSIDADE PAULISTA**

_____/___/___
**Prof. Richardson Luz**
**UNIVERSIDADE PAULISTA**

_____/___/___
**Prof. Elizeu Elieber**
**UNIVERSIDADE PAULISTA**

_____/___/___
**Prof. Waldir Antônio da Silva**
**UNIVERSIDADE PAULISTA**

## RESUMO

Este documento consiste em disponibilizar tanto o conceito como todos os procedimentos realizados para que o projeto venha a ter sucesso. O projeto aqui apresentado tem a finalidade de contribuir para que o aluno tenha uma comodidade a mais para ter acesso à ouvidoria da instituição, assim facilitando que a mesma solucione suas dúvidas de forma rápida e eficiente. Para que o projeto pudesse ser funcional, foi utilizado o ambiente de desenvolvimento *Visual Studio* onde foi criado todos os *layouts* de telas e escrito todo o código fonte com a linguagem de programação *CSharp*, assim permitindo que a aplicação final possa ser utilizada no sistema operacional *Windows,* além da criação do banco de dados em *SQL* com o *SQL Server Managment Studio* e com armazenar registros gerados pelo sistema. Com o uso deste projeto, estima-se que a universidade venha apresentar uma maior comodidade, portanto, gerando um resultado satisfatório tanto para os alunos como também para a universidade em si.

## ABSTRACT

This document consists of making available both the concept and all the procedures performed for the project to succeed. The project presented here has the purpose of contributing to the student having an additional convenience to have access to the institution's ombudsman, thus facilitating that the institution solve their doubts quickly and efficiently. In order for the project to be functional, we used the Visual Studio development environment where all the screen layouts were created and written all the source code with the CSharp programming language, thus allowing the final application to be used in the Windows operating system , in addition to creating the SQL database with SQL Server Management Studio and storing system-generated records. With the use of this project, it is estimated that the university will present a greater convenience, therefore, generating a satisfactory result both for the students as well as for the university itself.

**SUMÁRIO**

# 1. INTRODUÇÃO

O projeto apresenta um sistema que permite o aluno do campus ter acesso a ouvidoria da instituição de uma maneira mais pratica, com o sistema possuindo uma parte com dúvidas frequentes facilita com que o aluno não precise enviar uma dúvida simples, assim esclarecendo sua dúvida com mais agilidade, com isso a ouvidoria podendo focar em problemas de maior prioridade, além de ter um maior controle com a parte administrativa do sistema onde poderiam ver os registros das dúvidas enviadas.

## 1.1. Problema

A ouvidoria recebe apenas e-mails e não tem um registro dos mesmos em um banco de dados ou algo parecido, também recebem dúvidas iguais frequentemente, além de muitas vezes precisarem responder e-mails solicitando dados como RA ou CPF.

## 1.2. Hipótese

Auxiliar a ouvidoria da faculdade com um sistema que registre os e-mails enviados pelos alunos e evite com que o aluno envie um e-mail com falta de dados como por exemplo sem RA ou CPF, para que desta maneira acabe-se obtendo um controle melhor perante as dúvidas dos estudantes, além de que com isto pode-se ter um foco em dúvidas de maior prioridade.

## 1.3. Objetivos

### *1.3.1. Objetivos Gerais*

Desenvolver um sistema para desktop que auxilie a ouvidoria da faculdade.

### *1.3.2. Objetivos Específicos*

- Desenvolver a lógica de programação e as habilidades em formatar uma proposta de solução para o objetivo proposto;
- Aplicar e discutir as tecnologias utilizadas nos projetos de redes de computadores;
- Argumentar e discutir o uso de processos de software;
- Fomentar o hábito de trabalho em equipe e execução de projetos envolvendo múltiplas disciplinas.

## 1.4. Justificativa

Por conta dos fatores apresentados nos itens 1.1 e 1.2, se vê a necessidade de algo que venha auxiliar a ouvidoria, para que se tenham uma agilidade e foco melhor para solucionar dúvidas, além de que assim apresentem uma melhor comodidade aos alunos, demonstrando também uma boa qualidade de trabalho.

## 1.5. Metodologia

O presente documento tem como base metodológica os ensinamentos em sala de aula, pesquisas bibliográficas, manuais disponibilizados para o desenvolvimento.

## 2. O SISTEMA

A ideia do sistema surgiu após um dos autores do projeto enviar uma dúvida para a ouvidoria da instituição e obtendo como “resposta” a solicitação de seus dados, com isso surgiu a ideia de criação de algo para auxiliar a ouvidoria a evitar receber dúvidas com faltas de dados e outros problemas conforme já foi apresentado no projeto.

Com isso teve-se início a análise de requisitos que conforme (QUITERIO, 2012),

> A **Análise de Requisitos** ou **Engenharia de Requisitos** é um aspecto importante no Gerenciamento de Projetos, é a responsável por coletar dados indispensáveis, necessários, exigências de que o usuário necessite para solucionar um problema e alcançar seus objetivos. Assim como determinar as suas expectativas de um usuário para determinado produto.

Desta maneira utilizando a análise de requisitos o desenvolvimento do projeto seria melhor e mais prático além de apresentar menos problemas no decorrer do mesmo. Na análise de requisitos identificamos e reconhecemos o problema apresentado a cima no documento, com isso avaliamos o problema para desenvolvermos uma solução para o mesmo, definindo que seria necessário “dois sistemas”, o “primeiro” para o usuário no caso os alunos, onde seria apresentado duas opções de telas, uma que demonstrariam para o usuário dúvidas frequentes para que desta maneira evitasse o envio de duvidas simples do mesmo e poupasse tempo tanto do aluno como da ouvidoria, a outra interface apresentaria todo um formulário que seria preenchido pelo aluno onde evitaria a falta de dados no envio, o “segundo” sistema seria basicamente uma parte administrativa, onde haveria a tela de resposta utilizada pela ouvidoria, a partir disso avançamos para outra parte da análise que foi considerada como de maior prioridade, que seria a especificação dos requisitos, onde definimos como cada função seria executada e para o que seria, além de suas restrições, com isso para uma melhor visualização do sistema em si antes de seu desenvolvimento se viu necessário a utilização de itens UML que a partir de (RIBEIRO, 2012),

> UML é um acrônimo para a expressão Unified Modeling Language. Pela definição de seu nome, vemos que a UML é uma linguagem que define uma

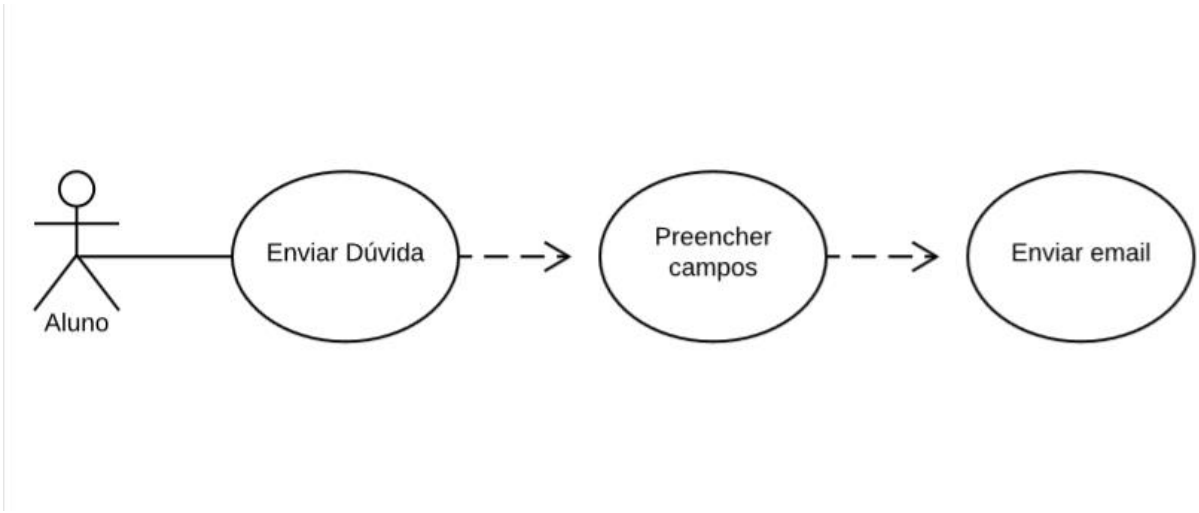
> série de artefatos que nos ajuda na tarefa de modelar e documentar os sistemas orientados a objetos que desenvolvemos.

## 2.1. UML

Inicialmente foi desenvolvido o diagrama de casos de uso que conforme (RIBEIRO, 2012),

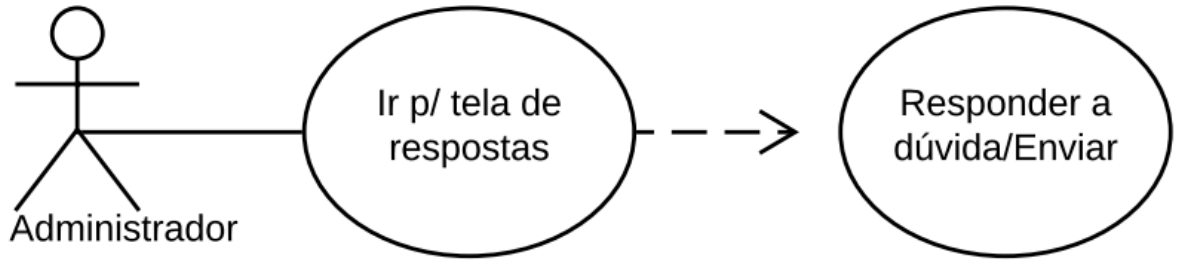
> Esse diagrama documenta o que o sistema faz do ponto de vista do usuário. Em outras palavras, ele descreve as principais funcionalidades do sistema e a interação dessas funcionalidades com os usuários do mesmo sistema. Nesse diagrama não nos aprofundamos em detalhes técnicos que dizem como o sistema faz.

Com isso temos a criação de um diagrama composto por quatro partes, cenário, ator, use case e comunicação, onde basicamente demonstra os eventos a partir da interação do usuário, também apresenta o usuário do sistema e suas funcionalidades, por fim a parte de comunicação serve para interligar um ator(usuário) com um caso de uso. Abaixo pode-se observar os diagramas de casos de uso para tela dos alunos e a tela administrativa:



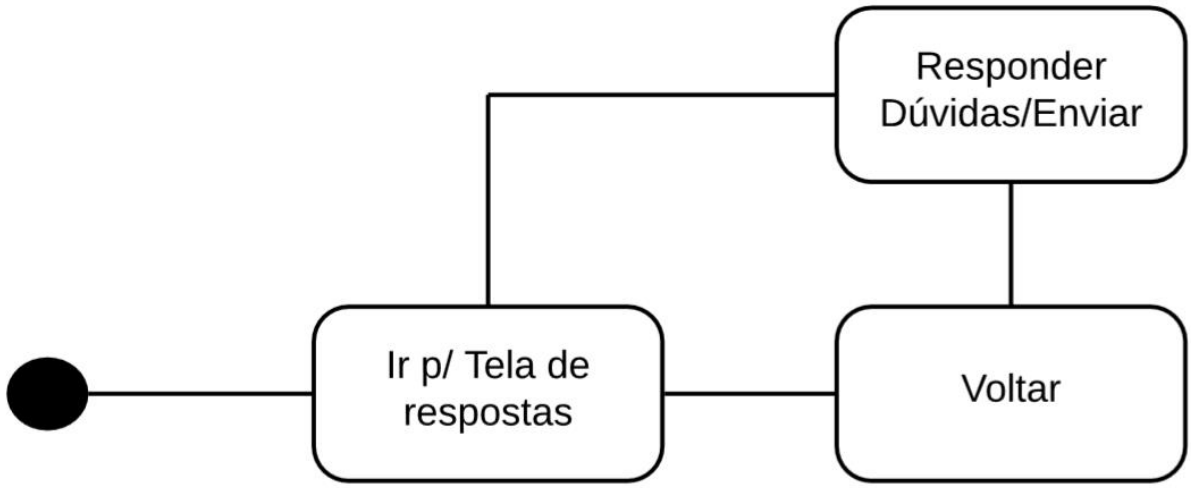

Fonte: Dos próprios autores

Fonte: Dos próprios autores

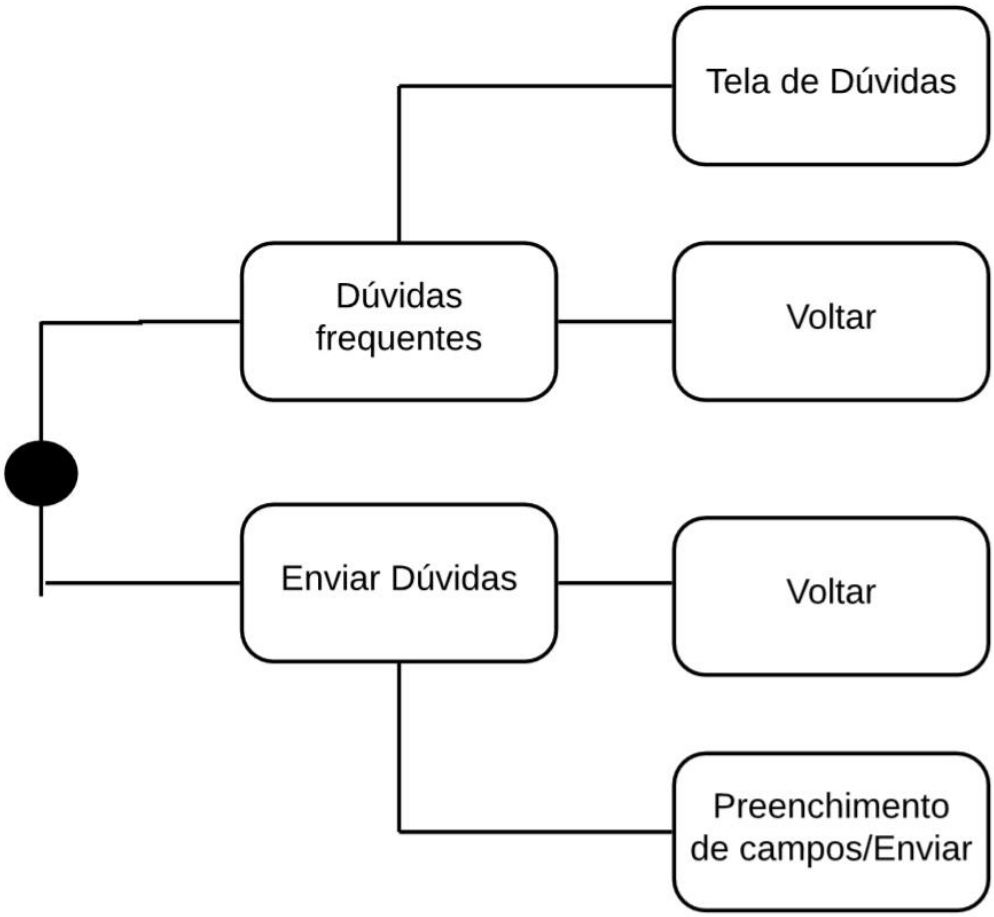



Fonte: Dos próprios autores

Após os diagramas de casos de uso, partimos para o desenvolvimento dos diagramas de sequência, que segundo (Guedes, 2011) “é um diagrama comportamental que preocupa-se com a ordem temporal em que as mensagens são trocadas entre os objetos envolvidos em um determinado processo“. Com o uso do diagrama de sequência temos o desenrolar dos eventos, assim dando um fim em toda esta parte de analises e partindo para o desenvolvimento do sistema em si, pode-se observar os diagramas de sequência apresentados abaixo para a tela de aluno e administrativa em seguida:




Fonte: Dos próprios autores




Fonte: Dos próprios autores

# 3. DESENVOLVIMENTO DO SISTEMA

Como o sistema seria orientado a objetos, para o desenvolvimento do mesmo foi decidido que a programação de todo o código fonte seria com a linguagem de programação C#Sharp por conta de ser voltada para o desenvolvimento orientado a objetos, a partir disso utilizamos o ambiente de desenvolvimento criado pela Microsoft chamado Visual Studio para escrever o código, além de desenvolver as telas dos sistemas e por fim compila-lo assim finalizando a parte do sistema. Obviamente que mesmo encerrando toda a parte da criação de *layout* de telas e desenvolvimento do código, o sistema não poderia funcionar totalmente, já que o foco principal seria o registro em um banco de dados, com isso foi criado um banco de dados, em SQL que segundo (Alves, 2013)

> é a linguagem padrão universal para manipular bancos de dados relacionais através dos SGBDs. Isso significa que todos os SGBDRs (Sistema de Gerenciamento de Banco de Dados Relacionais) oferecem uma interface para acessar o banco de dados utilizando a linguagem SQL, embora com algumas variações. Logo, saber o que é SQL e como utilizá-la é fundamental para qualquer desenvolvedor de softwares.

Com isso utilizando SQL Server Manegment Studio para criar o banco de dados e suas tabelas, com isso podendo finalizar o sistema por completo de forma funcional e com os objetivos atingidos.

## 3.1. Telas, classes, código fonte e padrões

No desenvolvimento do sistema utilizamos o padrão DAO que segundo (Dalepiane, 2014)

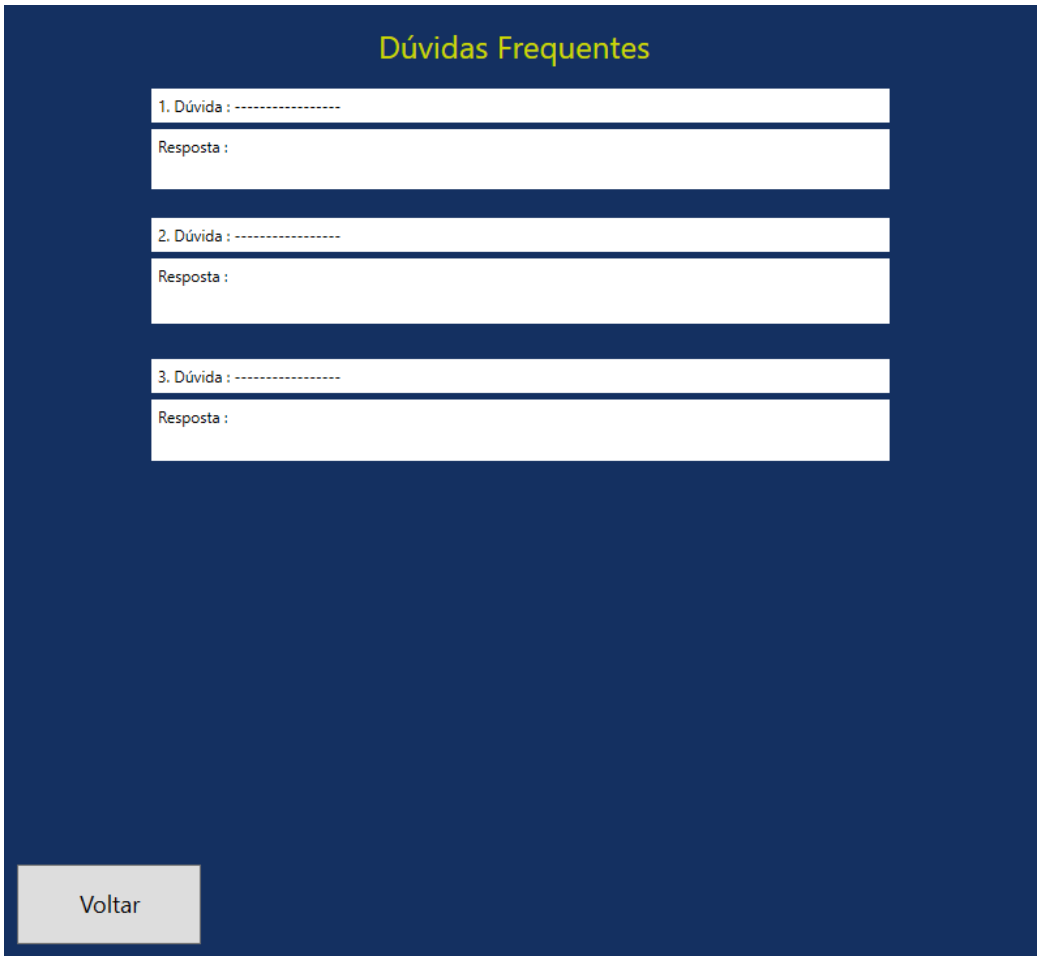
> O padrão de projeto DAO surgiu com a necessidade de separarmos a lógica de negócios da lógica de persistência de dados. Este padrão permite que possamos mudar a forma de persistência sem que isso influencie em nada na lógica de negócio, além de tornar nossas classes mais legíveis.

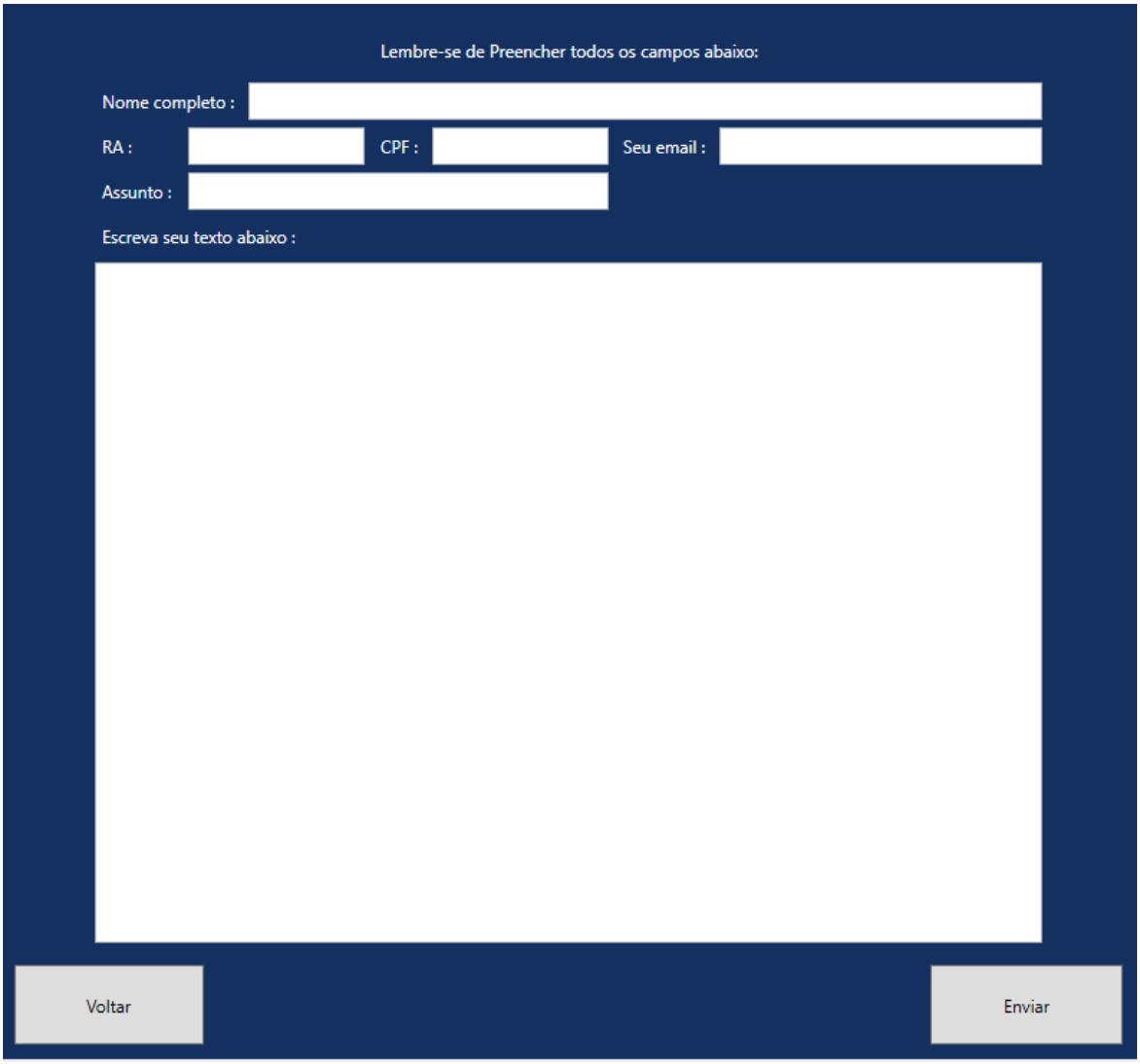
Assim podemos fazer a conexão do sistema com o banco de dados de forma que o sistema registre dados no banco sem problemas, como é um sistema orientado a objetos além do padrão DAO, há toda uma organização de classes e telas no desenvolvimento, basicamente são pastas que estão divididas em Modelo, Apresentação e DAL. Cada uma dessas divisão tem suas próprias classes e mesmo sendo “dois sistemas” a única divisão que há uma pequena mudança é a

Apresentação, onde no sistema para os alunos na pasta de Apresentação se tem o MainWindow.xaml que é a tela inicial conforme imagem abaixo:


Fonte: Dos próprios autores

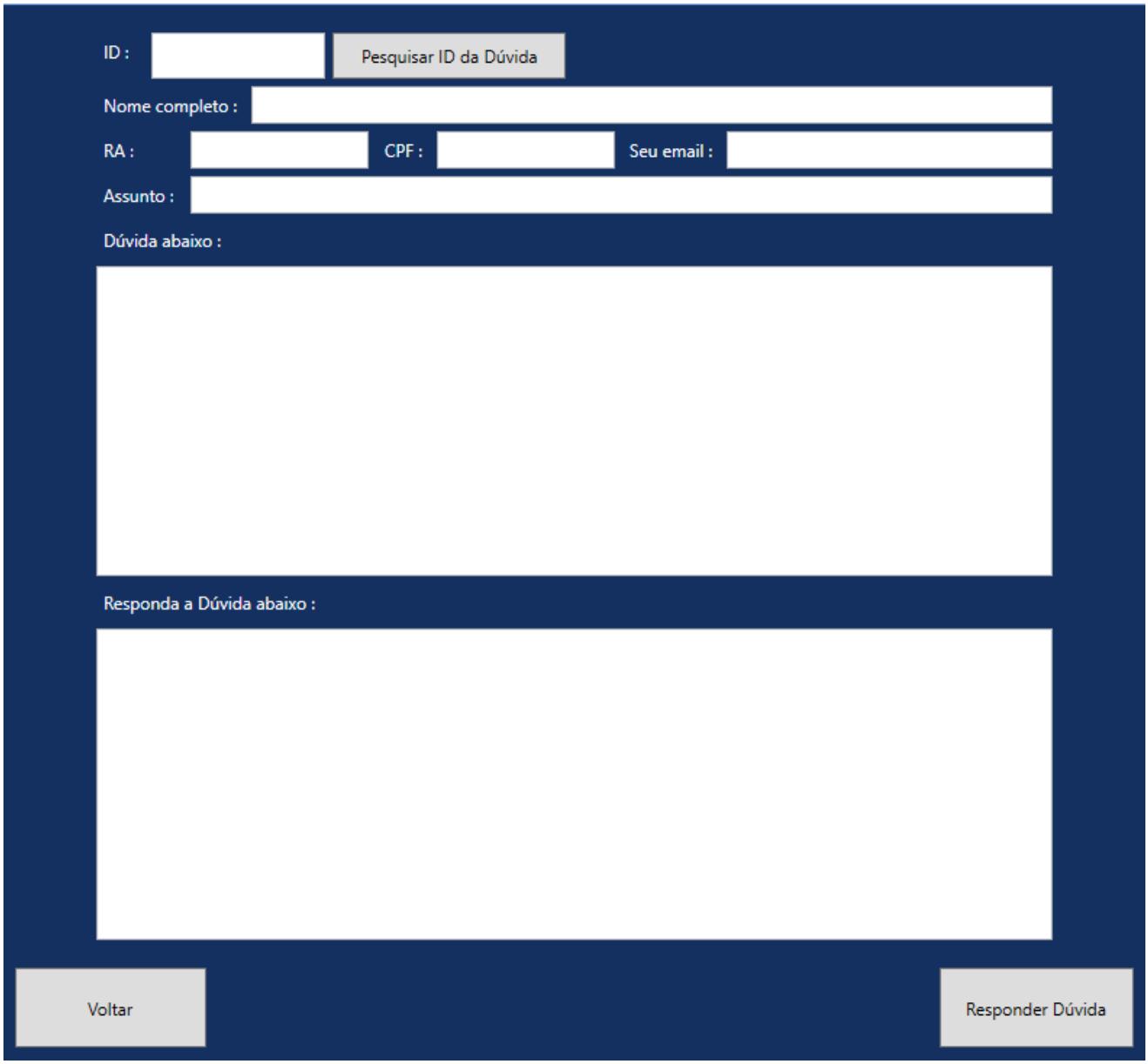
Continuando na apresentação no sistema dos alunos temos também os frmDuvidasFrequentes.xaml e frmJanelaDeEnvio.xaml, que seria as outras duas telas do sistema conforme demonstrados na mesma sequência abaixo:






Fonte: Dos próprios autores

Mesmo sendo apenas telas, por dentro delas também existe a parte de codificação, mas isso será deixado um pouco de lado e aparecerá mais para a frente, voltando a pasta de Apresentação a única diferença do sistema administrativo é que não a os frmDuvidasFrequentes.xaml e frmJanelaDeEnvio.xaml, mas sim o frmPCPI que é basicamente a tela de resposta para a ouvidoria utilizar conforme é demonstrado na imagem:




Fonte: Dos próprios autores

Agora, partindo para as próximas pastas, com relação aos dois sistemas, tanto o sistema dos alunos quanto o administrativo possuem as mesmas classes nas pastas Modelo e DAL, sendo eles na pasta Modelo, Controle.cs, Registros.cs, Validacao.cs e na pasta DAL temos Conexao.cs, intRegistrosDAO.cs e RegistrosDAO.cs, estas são classes apenas de códigos, basicamente o código é a única diferente que existe entre elas, sendo que algumas como por exemplo a classe Conexao.cs possui a mesma codificação nos dois sistema já que é apenas uma classe para a ligação com o banco de dados. A partir disto, desmontaremos todas os códigos fontes de cada classe e sua descrição para identificar a função de cada código apresentado.

### *3.1.1. Conexao.cs*

```
public class Conexao
{
    SqlConnection conexaoBD;

    public Conexao()
    {
        conexaoBD = new SqlConnection();
        conexaoBD.ConnectionString = @"Data Source=desktop-8328cgv\sqlexpress;
            Initial Catalog=ouvidoria;
            User ID=sa;Password=******";

    }
    //lembrar de colocar o usuário e senha

    public SqlConnection Conectar()
    {
        if (conexaoBD.State == System.Data.ConnectionState.Closed)
            conexaoBD.Open();
        return conexaoBD;
    }

    public void Desconectar()
    {
        if (conexaoBD.State == System.Data.ConnectionState.Open)
            conexaoBD.Close();
    }
}
```

Fonte: Dos próprios autores

Conforme já dito acima no documento a classe conexão serve apenas para ligar o sistema com o banco de dados, assim não tendo diferença nos dois sistemas.

### *3.1.2. intRegistrosDAO.cs e RegistrosDAO.cs*

```
interface intRegistrosDAO
{
    void EnviarRegistros(Registros registros);
}
```

Fonte: Dos próprios autores

```
public class RegistrosDAO : intRegistrosDAO
{
    Conexao conexaoBD = new Conexao();
    SqlDataReader dataReader;
    public String mensagem;
    public void EnviarRegistros(Registros registros)
    {
        this.mensagem = "";
        SqlCommand cmd = new SqlCommand();
        cmd.CommandText = @"insert into registrosD
            (nome, ra, cpf, assunto, email, texto)
            values
            (@nome, @ra, @cpf, @assunto, @email, @texto)";
        cmd.Parameters.AddWithValue("@nome", registros.nome);
        cmd.Parameters.AddWithValue("@ra", registros.ra);
        cmd.Parameters.AddWithValue("@cpf", registros.cpf);
        cmd.Parameters.AddWithValue("@assunto", registros.assunto);
        cmd.Parameters.AddWithValue("@email", registros.email);
        cmd.Parameters.AddWithValue("@texto", registros.texto);

        try
        {
            cmd.Connection = conexaoBD.Conectar();
            cmd.ExecuteNonQuery();
            conexaoBD.Desconectar();
            this.mensagem = "Dúvida enviada com sucesso !!!!!";
        }
        catch (SqlException e)
        {
            this.mensagem = e.ToString();
        }
    }
```

Fonte: Dos próprios autores

Os códigos acima representam as classes intRegistrosDAO.cs e RegistrosDAO.cs da pasta DAL do primeiro sistema no caso o dos alunos, a intRegistrosDAO é a classe de interface onde basicamente se compromete a fornecer um comportamento definido por esta interface, com isso temos o RegistrosDAO.cs que utiliza a classe de interface que definiu algo, como o padrão DAO é interligado com banco de dados, sendo assim na classe RegistrosDAO do sistema de alunos serve para enviar os registros para o banco de dados, conforme o usuário colocou nos campos da tela de envio das dúvidas. Abaixo podemos observar os códigos das mesmas classes, mas no sistema administrativo, pode-se perceber que as únicas diferenças são as funções enquanto no sistema de alunos os registros são envidados, no sistema administrativos os registros são carregados.

```
interface intRegistrosDAO
{
    Registros PesquisarPorID(Registros registros);
}
```

Fonte: Dos próprios autores

```
public class RegistrosDAO : intRegistrosDAO
{
    Conexao conexaoBD = new Conexao();
    SqlDataReader dataReader;
    public String mensagem;
    public Modelo.Registros PesquisarPorID(Modelo.Registros registros)
    {
        this.mensagem = "";
        SqlCommand cmd = new SqlCommand();
        cmd.CommandText = @"select * from registrosD
            where id = @id";
        cmd.Parameters.AddWithValue("@id", registros.id);
        try
        {
            cmd.Connection = conexaoBD.Conectar();
            dataReader = cmd.ExecuteReader();
            if (dataReader.HasRows)
            {
                dataReader.Read();
                registros.nome = dataReader["nome"].ToString();
                registros.ra = dataReader["ra"].ToString();
                registros.cpf = dataReader["cpf"].ToString();
                registros.assunto = dataReader["assunto"].ToString();
                registros.email = dataReader["email"].ToString();
                registros.texto = dataReader["texto"].ToString();
            }
            else
            {
                registros.id = 0;
            }
            dataReader.Close();
            conexaoBD.Desconectar();
        }
        catch (SqlException e)
        {
            this.mensagem = e.ToString();
        }
        return registros;
    }
}
```

Fonte: Dos próprios autores

### *3.1.3. Validacao.cs, Registros.cs e Controle.cs*

```
public class Validacao
{

    public String mensagem;
    public int id;

    public void ValidarDados(List<String> dadosRegistros)
    {
        this.mensagem = "";
        if (dadosRegistros[1].Length > 50)
            this.mensagem = "Nome com mais de 50 caracteres \n";
        if (dadosRegistros[2].Length > 7)
            this.mensagem += "RA com mais de 7 caracteres \n";
        if (dadosRegistros[3].Length > 13)
            this.mensagem += "CPF com mais de 13 caracteres \n";
        if (dadosRegistros[4].Length > 50)
            this.mensagem += "Assunto com mais de 50 caracteres, tenter ser mais direto \n";

        try
        {
            this.id = Convert.ToInt32(dadosRegistros[0]);
        }
        catch (FormatException e)
        {
            this.mensagem += "ID inválido";
        }
    }
}
```

Fonte: Dos próprios autores

A classe validação como o próprio nome já diz serve para validar itens, sendo assim impedindo e limitando o usuário a certas coisas, como na imagem acima impedindo que o usuário exceda um limite de caracteres que podem ser escritos em um campo.

```
public class Registros
{
    public int id { get; set; }
    public String nome { get; set; }
    public String ra { get; set; }
    public String cpf { get; set; }
    public String assunto { get; set; }
    public String email { get; set; }
    public String texto { get; set; }
    public String duvidas { get; set; }
}
```

Fonte: Dos próprios autores

Basicamente a classe Registros.cs define os objetos que serão utilizados com ligação com as telas e principalmente a pasta DAL, para desta maneira ligar com o banco de dados.

```
public class Controle
{
    public String mensagem;
    private object conexaoBD;

    public void EnivarRegistros(List<String> dadosRegistros)
    {
        this.mensagem = "";
        Validacao validacao = new Validacao();
        validacao.ValidarDados(dadosRegistros);
        if (validacao.mensagem.Equals(""))
        {
            Registros registros = new Registros();
            registros.nome = dadosRegistros[1];
            registros.ra = dadosRegistros[2];
            registros.cpf = dadosRegistros[3];
            registros.assunto = dadosRegistros[4];
            registros.email = dadosRegistros[5];
            registros.texto = dadosRegistros[6];
            DAL.RegistrosDAO registrosDAO = new DAL.RegistrosDAO();
            registrosDAO.EnviarRegistros(registros);
            this.mensagem = registrosDAO.mensagem;
        }
        else
        {
            this.mensagem = validacao.mensagem;
        }
    }
}
```

Fonte: Dos próprios autores

Controle.cs, esta classe tem uma ligação direta entre a Validacao.cs, RegistrosDAO.cs e as telas, basicamente como já se pode perceber esta classe interliga várias outras, sendo assim ela como o nome diz tem a função de controlar os eventos, recebendo os itens inseridos pelos usuários conforme os objetos definidos pela classe Registros.cs, utilizando a classe Validacao.cs para que nenhum limite seja ultrapassado e por fim utilizando enviando os registros para o banco de dados com a classe RegistrosDAO.cs, mas isso no sistema dos alunos, já no sistema administrativo como podemos observar na imagem abaixo basicamente faz quase as mesmas coisas, só que ao invés de enviar os itens faz com que os itens sejam recebidos do bando de dados.

```
public class Controle
{
    public String mensagem;
    public Modelo.Registros PesquisarPorID(List<String> dadosRegistros)
    {
        this.mensagem = "";
        Registros registros = new Registros();
        Validacao validacao = new Validacao();
        validacao.ValidarDados(dadosRegistros);
        if (validacao.mensagem.Equals(""))
        {
            registros.id = validacao.id;
            DAL.RegistrosDAO registrosDAO = new DAL.RegistrosDAO();
            registros = registrosDAO.PesquisarPorID(registros);
            this.mensagem = registrosDAO.mensagem;
        }
        else
        {
            this.mensagem = validacao.mensagem;
        }
        return registros;
    }
}
```

Fonte: Dos próprios autores

### *3.1.4. Funções das telas e seus códigos*

Como já é de se esperar cada tela tem uma função e como dito acima no documento, cada tela também tem seu código fonte, já que é necessário reconhecer cada *click* em algum botão pelo usuário ou algo digita em um campo pelo mesmo. Sendo assim as telas internamente tem seus códigos fontes, como veremos abaixo para a tela de envio das dúvidas que é utilizada pelos alunos temos os seguintes códigos:

```
    private void btnEnviar_Click(object sender, RoutedEventArgs e)
    {
        List<String> dadosRegistros = new List<string>();
        dadosRegistros.Add("0");
        dadosRegistros.Add(txbNome.Text);
        dadosRegistros.Add(txbRA.Text);
        dadosRegistros.Add(txbCPF.Text);
        dadosRegistros.Add(txbAssunto.Text);
        dadosRegistros.Add(txbEmail.Text);
        dadosRegistros.Add(txbTexto.Text);
        Modelo.Controle controle = new Modelo.Controle();
        controle.EnivarRegistros(dadosRegistros);
        MessageBox.Show(controle.mensagem);
    }
```

Fonte: Dos próprios autores

Como já foi dito antes e pode-se perceber no código acima a tela tem uma ligação com a classe Controle.cs, o código acima é uma função para um botão na tela, no caso o botão de envio, quando o usuário clica neste botão ele executa o código e pode-se notar que também são utilizados os campos de texto onde o usuário digita seus dados, como este código está ligado a classe controle, ele também tem todos os limites da validação e os campos de texto estão ligados aos objetos criados na classe Registros.cs, basicamente como é mostrado na imagem abaixo o código do sistema administrativo terá apenas as funções de carregar os registros do banco de dados a partir do *click* de um dos botões e por fim no *click* do último botão tem a função de enviar o e-mail de resposta para o aluno.

```
private void btnPPID_Click(object sender, RoutedEventArgs e)
{
    List<String> dadosRegistros = new List<string>();
    dadosRegistros.Add(txbID.Text);
    dadosRegistros.Add("");
    dadosRegistros.Add("");
    dadosRegistros.Add("");
    dadosRegistros.Add("");
    dadosRegistros.Add("");
    Modelo.Controle controle = new Modelo.Controle();
    Modelo.Registros registros = controle.PesquisarPorID(dadosRegistros);
    if (controle.mensagem.Equals(""))
    {
        txbNome.Text = registros.nome;
        txbRA.Text = registros.ra;
        txbCPF.Text = registros.cpf;
        txbAssunto.Text = registros.assunto;
        txbEmail.Text = registros.email;
        txbTexto.Text = registros.texto;
    }
    else
    {
        MessageBox.Show(controle.mensagem);
    }
}
```

Fonte: Dos próprios autores

```
private void btnResponder_Click(object sender, RoutedEventArgs e)
{
    SmtpClient usuario = new SmtpClient();
    NetworkCredential credenciais = new NetworkCredential();

    //definição de configurações do usuario
    usuario.Host = "smtp.gmail.com";
    usuario.Port = 587;
    usuario.EnableSsl = true;
    usuario.DeliveryMethod = SmtpDeliveryMethod.Network;
    usuario.UseDefaultCredentials = false;

    //definição de credenciais de acesso ao email
    credenciais.UserName = "emailparatestescsharp";
    credenciais.Password = "testeteste10";

    //definição das credenciais do usuario
    usuario.Credentials = credenciais;

    //preparar para enviar a mensagem
    MailMessage mensagem = new MailMessage();
    mensagem.From = new MailAddress("emailparatestescsharp@gmail.com");
    mensagem.Subject = (txbAssunto.Text);
    mensagem.IsBodyHtml = true;
    mensagem.Body = ("Olá aluno " + txbNome.Text + " Segue resposta da sua dúvida : <br><br>" + txbTextoR.Text);
    mensagem.To.Add(txbEmail.Text);

    //Envio do email
    try
    {
        usuario.Send(mensagem);
        MessageBox.Show("Resposta enviada com sucesso!");
    }
    catch (Exception)
    {
        MessageBox.Show("Não foi possivel enviar a resposta.");
    }

}
```

Fonte: Dos próprios autores

Então como é possível perceber todos os itens do sistema estão ligados, a Apresentação está ligada com o Modelo, com isso o Modelo liga com a DAL, por fim a DAL ligando o Bando de Dados externo. Tudo está interligado de certa forma, cada item necessita de outro para que o sistema funcione corretamente sem erros.

## 4. BANCO DE DADOS

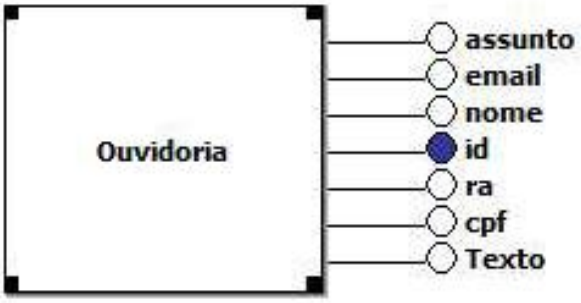
Um dos itens mais importantes e um dos mais importantes para a conclusão do objetivo principal é o banco de dados, pois como já foi dito desde a introdução um dos objetivos do projeto é registrar as dúvidas dos alunos para a ouvidoria ter um melhor controle. Como já foi apontado também o banco de dados foi desenvolvido em SQL com o software SQL Server Manegment Studio, mas aqui vamos demonstrar diferentes modelos de como o banco pode ser apresentado, no caso serão os modelos físico, conceitual e logico.

```
create database ouvidoria;

create table registroDuvidas(
        id int identity,
        nome varchar(100),
        ra varchar(7),
        cpf varchar(11),
        assunto varchar(100),
        email varchar(100),
        texto varchar(500),
        Primary Key (id)
);
```

Fonte: Dos próprios autores

A imagem acima representa o modelo físico do banco de dados, basicamente o código fonte, a partir deste código o banco pode ser criado no SQL Server Manegment Studio, assim criando também sua tabela, para posteriormente ser ligado junto do sistema e registrar os dados enviados por ele.




Fonte: Dos próprios autores

Conforme a imagem acima podemos ver o modelo conceitual, onde é apresentado de outra maneira o banco de dados e sua tabela, a partir do modelo conceitual é possível criar tanto o modelo físico como o modelo lógico já que ele tem o mais alto nível pois pode ser utilizado de maneira que envolva o cliente, pois se tem um item com uma facilidade no entendimento.


Fonte: Dos próprios autores

Podemos observar acima o modelo lógico tem uma função de implementar os recursos com adequação na sua nomenclatura, também definindo suas chaves primarias e suas chaves estrangeiras. Com isso temos todos os modelos do banco de dados, cada um deles apresenta uma maneira diferente de se ver o banco de dados em si, desta forma pode-se ter um entendimento melhor dependendo de qual modelo uma pessoa está vendo. Enfim basta apenas um banco de dados simples como este para poder auxiliar a ouvidoria da faculdade com os registros das dúvidas, com isso sempre haverá um jeito de acessar determinada dúvida de maneira mais rápida e pratica.

# CONCLUSÃO

Após todo o desenvolvimento do projeto, pode-se perceber vários aspectos necessários para o bom funcionamento do sistema e também o desenvolvimento do mesmo, como um dos principais o conhecimento das técnicas de programação com a linguagem CSharp para que as funções do sistema não erros na sua execução, obviamente não se pode deixar de lado o conhecimento na criação do bando de dados, pois teve grande importância no momento da criação e execução da parte administrativa do sistema. Um melhor controle foi obtido no desenvolvimento do sistema após utilizar alguns itens de UML como diagramas de caso de uso e sequencia, além de um diagrama de classes, com isso possibilitando uma visão melhor do sistema e se tendo uma noção aprimorada de quais alterações seriam necessárias para a conclusão do projeto.

# REFERÊNCIAS

ALVES, Gustavo Furtado de Oliveira. **Você precisa saber o que é SQL!** 2013. Disponível em: < https://dicasdeprogramacao.com.br/o-que-e-sql/>. Acesso em: 23 outubro de 2018.

DALEPIANE, Felipe. **DAO Pattern: Persistência de Dados utilizando o padrão DAO.** 2014. Disponível em: < https://www.devmedia.com.br/dao-pattern-persistencia-de-dados-utilizando-o-padrao-dao/30999>. Acesso em:

GUEDES, Gilleanes T. A. **UML 2. Uma abordagem prática.** 2. ed. São Paulo: Novatec Editora Ltda. 2011. 30 p.

QUITERIO, Ana Paula. **Analise de Requisitos**. 2012. Disponível em: <https://www.infoescola.com/engenharia-de-software/analise-de-requisitos/>. Acesso em: 15 de setembro de 2018.

RIBEIRO, Leandro. **O que é UML e Diagramas de Caso de Uso**: Introdução Prática à UML. 2012. Disponível em: < https://www.devmedia.com.br/o-que-e-uml-e-diagramas-de-caso-de-uso-introducao-pratica-a-uml/23408>. Acesso em: 3 de outubro de 2018.