```
public class ConviteDomain
{
    public int Id { get; set; }
    public int EventoId { get; set; }
    public EventoDomain Evento { get; set; }

    public int UsuarioId { get; set; }
    public UsuarioDomain Usuario { get; set; }

    public EnSituacaoConvite Situacao { get; set; }
}
```

1. API
2. Startup – configurar o MVC e a versão que eu estou utilizando

3. TipoEventoController
   TipoEventoDomain
4. ITipoEventoRepository
5. TipoEventoRepository

6. InstituicaoController
7. InstituicaoDomain
8. IInstituicaoRepository
9. InstituicaoRepository

10. Validar dados de entrada com anotação

11. UsuarioDomain
12. UsuariosController
13. IUsuarioRepository
14. UsuarioRepository

15. JWT – autenticação
16. Ajustes dos endpoints para barrar o acesso

17. EventoDomain
18. EventosController
19. IEventoRepository

20. EventoRepository

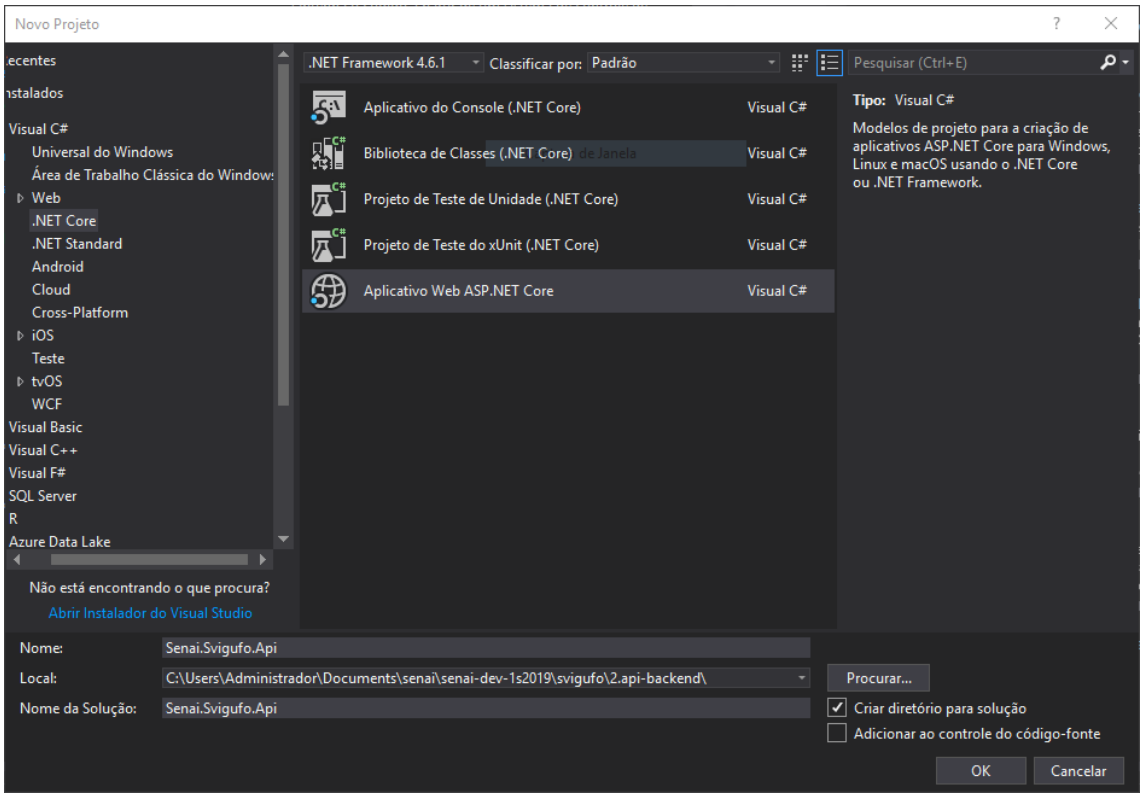
21. ConviteDomain
22. ConvitesController
23. IConviteRepository
24. ConviteRepository

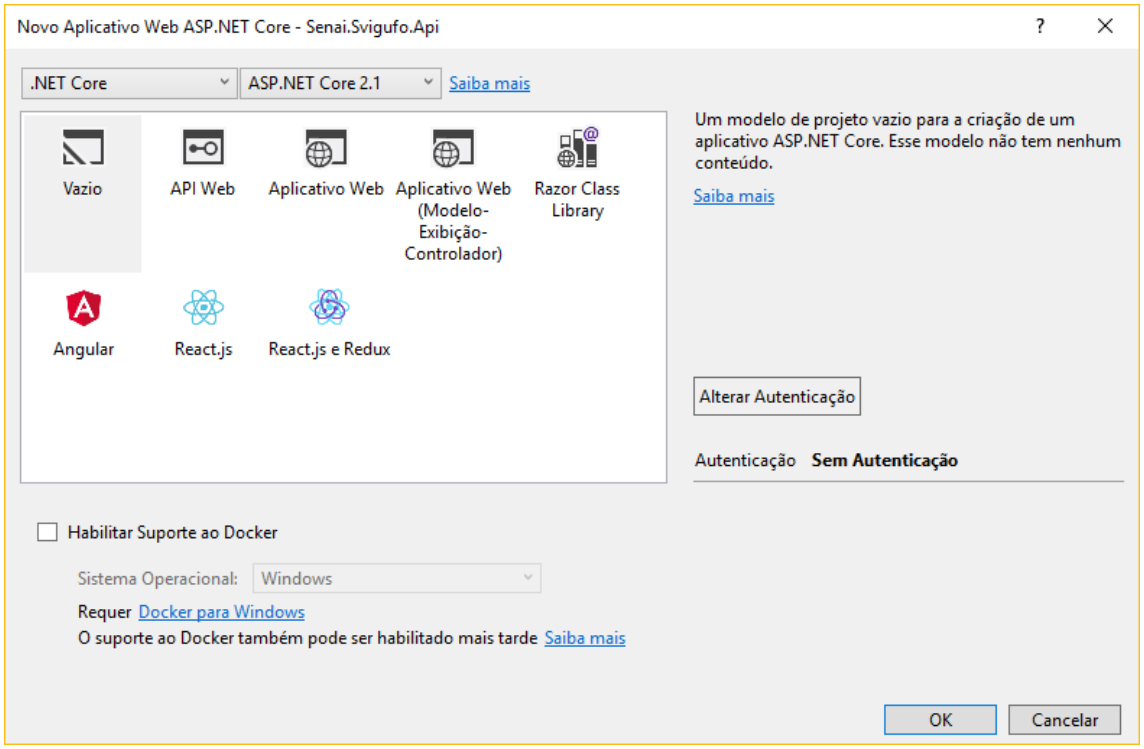
Explicar todo o contexto de API com imagens e apresentações.

Explicar o que é JSON, API, vantagens

Somente após explicar todo o contexto, mostrar um projeto sendo criado

Criar um novo projeto WebApi

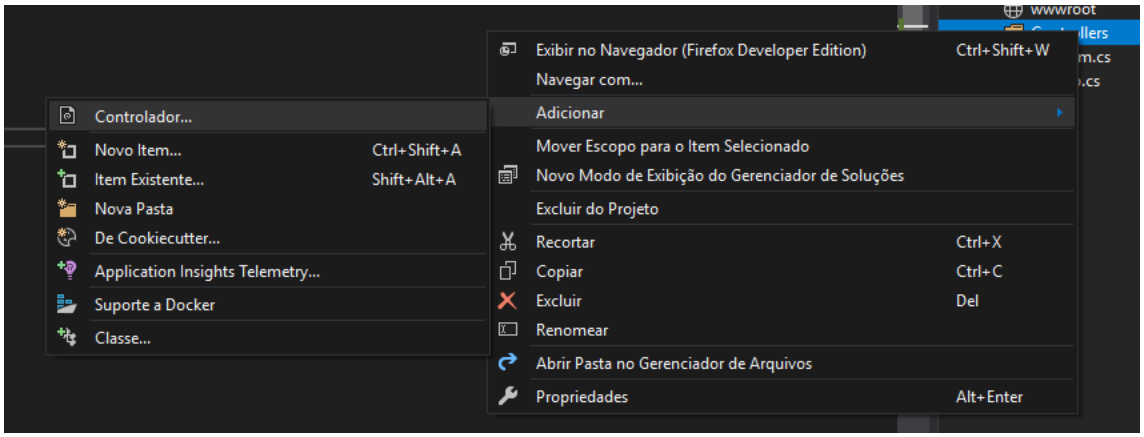





```
1 referência
public class Startup
{
    // This method gets called by the runtime. Use this method to add services to the container.
    // For more information on how to configure your application, visit https://go.microsoft.com/fwlink/?LinkID=398940
    0 referências | 0 exceções
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddMvc().SetCompatibilityVersion(Microsoft.AspNetCore.Mvc.CompatibilityVersion.Version_2_1);
    }
```

```
// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
0 referências | 0 exceções
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    app.UseMvc();

    //app.Run(async (context) =>
    //{
    //    await context.Response.WriteAsync("Hello World!");
    //});
}
```

Nova pasta Controllers



QUANDO CRIAR O CONTROLLER, LEMBRAR DE FALAR SOBRE O CONTROLLERBASE – QUE ATUA SEM O SUPORTE A VIEW, POIS NÃO IREMOS PRECISAR DELA

TiposEventosController

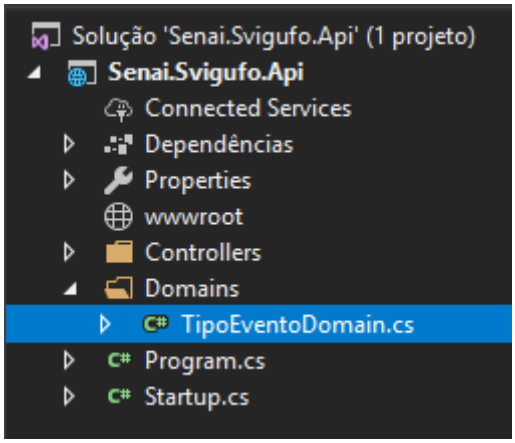
```
[Route("api/[controller]")]
0 referências
public class TiposEventosController : Controller
{

    [HttpGet]
    0 referências | 0 solicitações | 0 exceções
    public string Get()
    {
        return "Requisição Recebida com Sucesso.";
    }

}
```

Realizar requisição

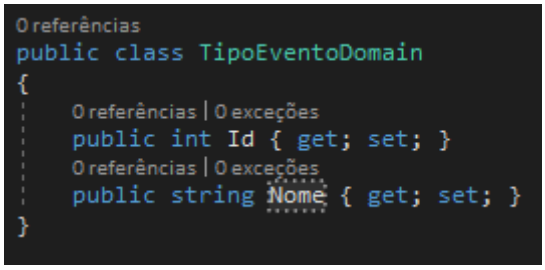
Mostrar na web – Navegador, a informação recebida – Não iniciar com o postman ainda

Criar nova pasta – Domains

Criar nova classe TipoEventoDomain



Incluir as propriedades



https://stackoverflow.com/questions/23196931/returning-ihttpactionresult-vs-ienumerableitem-vs-iqueryableitem

Diferença entre IEnumerable e IActionResult

Realizar um retorno simples com uma lista local

```
List<TipoEventoDomain> eventos = new List<TipoEventoDomain>() {
    new TipoEventoDomain { Id = 1, Nome = "Tipo Evento A" }
    ,new TipoEventoDomain { Id = 2, Nome = "Tipo Evento B" }

};

//[HttpGet]
//public string Get()
//{
//    return "Requisição Recebida com Sucesso.";
//}

[Produces("application/json")]
[HttpGet]
0 referências | 0 solicitações | 0 exceções
public IEnumerable<TipoEventoDomain> Get()
{
    return eventos;
}
```

Realizar a requisição

```
[Produces("application/json")]
[HttpGet]
0 referências | 0 solicitações | 0 exceções
public IEnumerable<TipoEventoDomain> Get()
{
    return eventos;
}
```

LEMBRAR DE INCLUIR O CORS NA NOSSA APLICAÇÃO PARA MOSTRAR O FRONT

```
namespace Senai.Svigufo.Api
{
    1 referência
    public class Startup
    {
        0 referências | 0 exceções
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddMvc().SetCompatibilityVersion(Microsoft.AspNetCore.Mvc.CompatibilityVersion.Version_2_1);

            services.AddCors(options =>
            {
                options.AddPolicy("CorsPolicy",
                    builder => builder.AllowAnyOrigin()
                        .AllowAnyMethod()
                        .AllowAnyHeader()
                        .AllowCredentials());
            });
        }

        0 referências | 0 exceções
        public void Configure(IApplicationBuilder app, IHostingEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }

            app.UseCors(builder => builder.AllowAnyHeader().AllowAnyMethod().AllowAnyOrigin());

            app.UseMvc();

        }
    }
}
```

Mostrar o front-end fazendo a requisição para o back-end

E mostrar que sua implementação não é conhecida

Eu nem mesmo sei/conheço aonde esses dados estão sendo armazenados

Adicionar mais dois eventos e mostrar o resultado

E também mostrar a interação no front-end

```
List<TipoEventoDomain> eventos = new List<TipoEventoDomain>() {
    new TipoEventoDomain { Id = 1, Nome = "Tipo Evento A" }
    ,new TipoEventoDomain { Id = 2, Nome = "Tipo Evento B" }
    ,new TipoEventoDomain { Id = 3, Nome = "Tipo Evento C" }
    ,new TipoEventoDomain { Id = 4, Nome = "Tipo Evento D" }
};

//[HttpGet]
//public string Get()
//{
//    return "Requisição Recebida com Sucesso.";
//}

[Produces("application/json")]
[HttpGet]
0 referências | 0 solicitações | 0 exceções
public IEnumerable<TipoEventoDomain> Get()
{
    return eventos;
}
```

Buscando somente um

```
List<TipoEventoDomain> eventos = new List<TipoEventoDomain>() {
    new TipoEventoDomain { Id = 1, Nome = "Tipo Evento A" }
    ,new TipoEventoDomain { Id = 2, Nome = "Tipo Evento B" }
    ,new TipoEventoDomain { Id = 3, Nome = "Tipo Evento C" }
    ,new TipoEventoDomain { Id = 4, Nome = "Tipo Evento D" }
};

//[HttpGet]
//public string Get()
//{
//    return "Requisição Recebida com Sucesso.";
//}

[Produces("application/json")]
[HttpGet]
0 referências | 0 solicitações | 0 exceções
public IEnumerable<TipoEventoDomain> Get()
{
    return eventos;
}

[HttpGet("{id}")]
0 referências | 0 solicitações | 0 exceções
public IActionResult GetById(int id)
{
    return Ok(eventos.Find(x => x.Id == id));
}
```

Melhorar realizando um if para ver se o evento foi encontrado

```
List<TipoEventoDomain> eventos = new List<TipoEventoDomain>() {
    new TipoEventoDomain { Id = 1, Nome = "Tipo Evento A" }
    ,new TipoEventoDomain { Id = 2, Nome = "Tipo Evento B" }
    ,new TipoEventoDomain { Id = 3, Nome = "Tipo Evento C" }
    ,new TipoEventoDomain { Id = 4, Nome = "Tipo Evento D" }
};

//[HttpGet]
//public string Get()
//{
//    return "Requisição Recebida com Sucesso.";
//}

[Produces("application/json")]
[HttpGet]
0 referências | 0 solicitações | 0 exceções
public IEnumerable<TipoEventoDomain> Get()
{
    return eventos;
}

[HttpGet("{id}")]
0 referências | 0 solicitações | 0 exceções
public IActionResult GetById(int id)
{
    TipoEventoDomain evento = eventos.Find(x => x.Id == id);
    if (evento == null)
    {
        return NotFound();
    }
    return Ok(evento);
}
```

```
[Route("api/[controller]")]
0 referências
public class TiposEventosController : ControllerBase
{

    List<TipoEventoDomain> eventos = new List<TipoEventoDomain>() {
        new TipoEventoDomain { Id = 1, Nome = "Tipo Evento A" }
        ,new TipoEventoDomain { Id = 2, Nome = "Tipo Evento B" }
        ,new TipoEventoDomain { Id = 3, Nome = "Tipo Evento C" }
        ,new TipoEventoDomain { Id = 4, Nome = "Tipo Evento D" }
    };

    //[HttpGet]
    //public string Get()
    //{
    //    return "Requisição Recebida com Sucesso.";
    //}

    [Produces("application/json")]
    [HttpGet]
    0 referências | 0 solicitações | 0 exceções
    public IEnumerable<TipoEventoDomain> Get()
    {
        return eventos;
    }

    [Produces("application/json")]
    [HttpGet("{id}")]
    0 referências | 0 solicitações | 0 exceções
    public IActionResult GetById(int id)
    {
        TipoEventoDomain evento = eventos.Find(x => x.Id == id);
        if (evento == null)
        {
            return NotFound();
        }
        return Ok(evento);
    }
```

A saída também será produces

Podendo assim, colocar no controller diretamente

```
[Produces("application/json")]
[Route("api/[controller]")]
0 referências
public class TiposEventosController : ControllerBase
{

    List<TipoEventoDomain> eventos = new List<TipoEventoDomain>() {
        new TipoEventoDomain { Id = 1, Nome = "Tipo Evento A" }
        ,new TipoEventoDomain { Id = 2, Nome = "Tipo Evento B" }
        ,new TipoEventoDomain { Id = 3, Nome = "Tipo Evento C" }
        ,new TipoEventoDomain { Id = 4, Nome = "Tipo Evento D" }
    };

    //[HttpGet]
    //public string Get()
    //{
    //    return "Requisição Recebida com Sucesso.";
    //}

    [HttpGet]
    0 referências | 0 solicitações | 0 exceções
    public IEnumerable<TipoEventoDomain> Get()
    {
        return eventos;
    }

    [HttpGet("{id}")]
    0 referências | 0 solicitações | 0 exceções
    public IActionResult GetById(int id)
    {
        TipoEventoDomain evento = eventos.Find(x => x.Id == id);
        if (evento == null)
        {
            return NotFound();
        }
        return Ok(evento);
    }
```

Trabalhando com POST

MOSTRAR SOMENTE AQUI O POSTMAN

```
[HttpPost]
0 referências | 0 solicitações | 0 exceções
public IActionResult Post([FromBody] TipoEventoDomain tipoEvento) {

    //eventos.Add(new TipoEventoDomain() { Id = eventos.Count + 1, Nome = tipoEvento.Nome });
    //return Ok(eventos);
    tipoEvento.Id = eventos.Count + 1;
    return Ok(tipoEvento);
}
```

Explicar a anotação [ApiController]

```
[Produces("application/json")]
[Route("api/[controller]")]
[ApiController]
0 referências
public class TiposEventosController : ControllerBase
{

    List<TipoEventoDomain> eventos = new List<TipoEventoDomain>() {
        new TipoEventoDomain { Id = 1, Nome = "Tipo Evento A" }
        ,new TipoEventoDomain { Id = 2, Nome = "Tipo Evento B" }
        ,new TipoEventoDomain { Id = 3, Nome = "Tipo Evento C" }
        ,new TipoEventoDomain { Id = 4, Nome = "Tipo Evento D" }
    };

    //[HttpGet]
    //public string Get()
    //{
    //    return "Requisição Recebida com Sucesso.";
    //}

    [HttpGet]
    0 referências | 0 solicitações | 0 exceções
    public IEnumerable<TipoEventoDomain> Get()...

    [HttpGet("{id}")]
    0 referências | 0 solicitações | 0 exceções
    public IActionResult GetById(int id)...

    [HttpPost]
    0 referências | 0 solicitações | 0 exceções
    public IActionResult Post(TipoEventoDomain tipoEvento) {

        //eventos.Add(new TipoEventoDomain() { Id = eventos.Count + 1, Nome = tipoEvento.Nome });
        //return Ok(eventos);
        tipoEvento.Id = eventos.Count + 1;
        return Ok(tipoEvento);
    }

}
```

Mostrar o PUT

```
[HttpPut]
0 referências | 0 solicitações | 0 exceções
public IActionResult Put(TipoEventoDomain tipoEvento) {
    var eventoEncontrado = eventos.Find(x => x.Id == tipoEvento.Id);
    eventoEncontrado.Nome = tipoEvento.Nome;

    return Ok(eventos);
}
```

DELETE

```
[HttpDelete("{id}")]
0 referências | 0 solicitações | 0 exceções
public IActionResult Delete(int id) {
    eventos.Remove(eventos.Find(x => x.Id == id));
    return Ok(eventos);
}
```

INCLUIR COMENTÁRIOS NO CÓDIGO

```
/// <summary>
/// Busca somente um tipo de evento cadastrado
/// </summary>
/// <param name="id">Id</param>
/// <returns>Retorna um tipo de evento encontrado</returns>
[HttpGet("{id}")]
0 referências | 0 solicitações | 0 exceções
public IActionResult GetById(int id)...

/// <summary>
/// Cadastrar um novo tipo de evento
/// </summary>
/// <param name="tipoEvento"></param>
/// <returns>Retorna a lista atualizada</returns>
[HttpPost]
0 referências | 0 solicitações | 0 exceções
public IActionResult Post(TipoEventoDomain tipoEvento)
{

    eventos.Add(new TipoEventoDomain() { Id = eventos.Count + 1, Nome = tipoEvento.Nome });
    return Ok(eventos);
    //tipoEvento.Id = eventos.Count + 1;
    //return Ok(tipoEvento);
}
```

BANCO DE DADOS

Criar uma nova interface

```
/// <summary>
/// Interface responsável pelo Tipo de Evento Repository
/// </summary>
public interface ITipoEventoRepository
{
    void Cadastrar(TipoEventoDomain tipoEvento);
    void Alterar(TipoEventoDomain tipoEvento);
    TipoEventoDomain BuscarPorId(int id);
    List<TipoEventoDomain> Listar();
    #region Deleção
    void Deletar(int id);
    #endregion
}
```

Criar um novo repositório

```
public class TipoEventoRepository : ITipoEventoRepository
{

    public List<TipoEventoDomain> Listar()...

    public void Alterar(TipoEventoDomain tipoEvento)...

    public TipoEventoDomain BuscarPorId(int id)...

    public void Cadastrar(TipoEventoDomain tipoEvento)...

    public void Deletar(int id)...
}
```

No controller, vamos alterar para ao invés de trabalhar com a lista local, chamarmos do nosso banco de dados.

Cria o construtor no controller

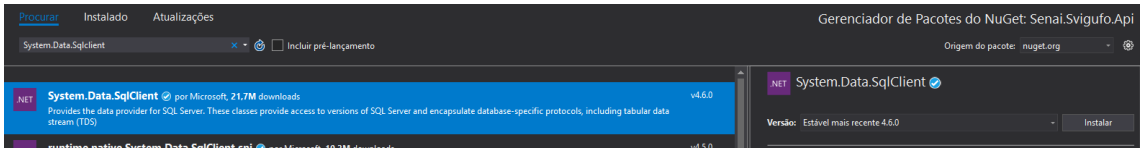
```
1 referência | 0 exceções
private ITipoEventoRepository TipoEventoRepositorio { get; set; }

0 referências | 0 exceções
public TiposEventosController()
{
    TipoEventoRepositorio = new TipoEventoRepository();
}

List<TipoEventoDomain> eventos = new List<TipoEventoDomain>()...;

//[HttpGet]
//public string Get()
//{
//    return "Requisição Recebida com Sucesso.";
//}

/// <summary>
/// Retorna a lista de tipos de eventos
/// </summary>
/// <returns>
/// Retorna a lista de tipos de eventos
/// </returns>
[HttpGet]
0 referências | 0 solicitações | 0 exceções
public IEnumerable<TipoEventoDomain> Get()
{
    return eventos;
}
```

Todo a sua implementação continua a mesma

```
/// <summary>
/// Retorna a lista de tipos de eventos
/// </summary>
/// <returns>
/// Retorna a lista de tipos de eventos
/// </returns>
[HttpGet]
0 referências | 0 solicitações | 0 exceções
public IEnumerable<TipoEventoDomain> Get()
{
    return TipoEventoRepositorio.Listar();
}
```

Porém, os dados virão agora do banco de dados

Botão direito -> gerenciar pacotes do nuget



ADO.NET

https://docs.microsoft.com/pt-br/dotnet/framework/data/adonet/

- ExecuteScalar is typically used when your query returns a single value. If it returns more, then the result is the first column of the first row. An example might be SELECT @@IDENTITY AS 'Identity'.
- ExecuteReader is used for any result set with multiple rows/columns (e.g., SELECT col1, col2 from sometable).
- ExecuteNonQuery is typically used for SQL statements without results (e.g., UPDATE, INSERT, etc.).

Implementação do Repositório

LISTAGEM

```
private string stringDeConexao = "Data Source=localhost;Initial Catalog=SENAI_SVIGUFO;Integrated Security=True";
private string queryASerExecutada = "SELECT ID, TITULO FROM TIPOS_EVENTOS";

public List<TipoEventoDomain> Listar()
{
    // crio uma nova lista
    List<TipoEventoDomain> listaTiposEventos = new List<TipoEventoDomain>();

    // garante que os recursos sejam fechados e descartados quando o código será encerrado
    // sqlconnection cria uma instância do objeto
    using (SqlConnection con = new SqlConnection(stringDeConexao))
    {
        // permite que você crie queries e envie para o banco de dados
        using (SqlCommand cmd = new SqlCommand(queryASerExecutada, con))
        {
            // para determinar que é do tipo texto
            // cmd.CommandType = CommandType.Text;
            con.Open();
            // retorna um conjunto de resultados do banco de dados
            SqlDataReader rdr = cmd.ExecuteReader();
            // enquanto tiver registros
            while (rdr.Read())
            {
                // cria um novo objeto e insere na lista
                TipoEventoDomain tipoEvento = new TipoEventoDomain
                {
                    Id = Convert.ToInt32(rdr["ID"]),
                    Nome = rdr["TITULO"].ToString()
                };

                // adiciona cada evento na lista
                listaTiposEventos.Add(tipoEvento);
            }
        }
    }
    // retorna a própria lista
    return listaTiposEventos;
}
```

BUSCAR SOMENTE UM

```
public TipoEventoDomain BuscarPorId(int id)
{

    string queryASerExecutada = "SELECT ID, TITULO FROM TIPOS_EVENTOS WHERE ID = @ID";

    // crio um novo evento
    TipoEventoDomain tipoEvento = new TipoEventoDomain();
    // abro uma nova instância na conexão
    using (SqlConnection con = new SqlConnection(stringDeConexao))
    {
        // um novo comando que será executado na conexão
        using (SqlCommand cmd = new SqlCommand(queryASerExecutada, con))
        {
            // preciso passar o parâmetro que recebi, para a query a ser executada
            cmd.Parameters.AddWithValue("@ID", id);
            // abre a conexão
            con.Open();
            // lê os registros
            SqlDataReader lerOsRegistros = cmd.ExecuteReader();
            // mas agora eu posso verificar se tenho linhas
            if (lerOsRegistros.HasRows)
            {
                // enquanto eu tiver registros, no caso, somente um, coloco os dados no objeto criado
                while (lerOsRegistros.Read())
                {
                    tipoEvento.Id = Convert.ToInt32(lerOsRegistros["ID"].ToString());
                    tipoEvento.Nome = lerOsRegistros["TITULO"].ToString();
                }
                return tipoEvento;
            }
        }
    }

    // caso não tenha registro, o valor será nulo
    return null;
}
```

Mudo a implementação no controller, para realizar a busca agora no repositório

```
/// <summary>
/// Busca somente um tipo de evento cadastrado
/// </summary>
/// <param name="id">Id</param>
/// <returns>Retorna um tipo de evento encontrado</returns>
[HttpGet("{id}")]
public IActionResult GetById(int id)
{
    // TipoEventoDomain evento = eventos.Find(x => x.Id == id);
    TipoEventoDomain evento = TipoEventoRepositorio.BuscarPorId(id);
    if (evento == null)
    {
        return NotFound();
    }
    return Ok(evento);
}
```

Alterar o repositório para cadastrar um novo tipo de evento

```
public void Cadastrar(TipoEventoDomain tipoEvento)
{
    using (SqlConnection con = new SqlConnection(stringDeConexao))
    {
        // string queryASerExecutada = "INSERT INTO TIPOS_EVENTOS (TITULO) VALUES (' " + tipoEvento.Nome + " ')";
        string queryASerExecutada = "INSERT INTO TIPOS_EVENTOS (TITULO) VALUES (@TITULO)";

        SqlCommand cmd = new SqlCommand(queryASerExecutada, con);
        cmd.Parameters.AddWithValue("@TITULO", tipoEvento.Nome);
        con.Open();
        cmd.ExecuteNonQuery();
    }
}
```

Alterar o controller

```
/// <summary>
/// Cadastrar um novo tipo de evento
/// </summary>
/// <param name="tipoEvento"></param>
/// <returns>Retorna a lista atualizada</returns>
[HttpPost]
public IActionResult Post(TipoEventoDomain tipoEvento)
{

    //eventos.Add(new TipoEventoDomain() { Id = eventos.Count + 1, Nome = tipoEvento.Nome });
    //return Ok(eventos);
    //tipoEvento.Id = eventos.Count + 1;
    //return Ok(tipoEvento);
    TipoEventoRepositorio.Cadastrar(tipoEvento);
    return Ok();
}
```

Realizar uma verificação na inserção para não travar a inserção

```
/// <summary>
/// Cadastrar um novo tipo de evento
/// </summary>
/// <param name="tipoEvento"></param>
/// <returns>Retorna a lista atualizada</returns>
[HttpPost]
public IActionResult Post(TipoEventoDomain tipoEvento)
{

    //eventos.Add(new TipoEventoDomain() { Id = eventos.Count + 1, Nome = tipoEvento.Nome });
    //return Ok(eventos);
    //tipoEvento.Id = eventos.Count + 1;
    //return Ok(tipoEvento);
    try
    {
        TipoEventoRepositorio.Cadastrar(tipoEvento);
        return Ok();
    }
    catch
    {
        return BadRequest();
    }
}
```

Alterar a implementação no repositório para atualizar

```
public void Alterar(TipoEventoDomain tipoEvento)
{
    using (SqlConnection con = new SqlConnection(stringDeConexao))
    {
        string queryASerExecutada = "UPDATE TIPOS_EVENTOS SET TITULO = @TITULO WHERE ID = @ID;";
        SqlCommand cmd = new SqlCommand(queryASerExecutada, con);
        cmd.Parameters.AddWithValue("@TITULO", tipoEvento.Nome);
        cmd.Parameters.AddWithValue("@ID", tipoEvento.Id);

        con.Open();
        cmd.ExecuteNonQuery();
    }
}
```

Ajustar a implementação no controller

```
/// <summary>
/// Atualiza um tipo de evento
/// </summary>
/// <param name="tipoEvento"></param>
/// <returns>Retorna a lista de atualizada de tipos de eventos</returns>
[HttpPut]
public IActionResult Put(TipoEventoDomain tipoEvento)
{
    //var eventoEncontrado = eventos.Find(x => x.Id == tipoEvento.Id);
    //eventoEncontrado.Nome = tipoEvento.Nome;

    //return Ok(eventos);
    try
    {
        TipoEventoRepositorio.Alterar(tipoEvento);
        return Ok();
    }
    catch
    {
        return BadRequest();
    }
}
```

E caso o meu tipo de evento não seja encontrado, posso retornar um valor não encontrado.

```
/// <summary>
/// Atualiza um tipo de evento
/// </summary>
/// <param name="tipoEvento"></param>
/// <returns>Retorna a lista de atualizada de tipos de eventos</returns>
[HttpPut]
public IActionResult Put(TipoEventoDomain tipoEvento)
{
    //var eventoEncontrado = eventos.Find(x => x.Id == tipoEvento.Id);
    //eventoEncontrado.Nome = tipoEvento.Nome;

    //return Ok(eventos);

    TipoEventoDomain eventoASerAtualizado = TipoEventoRepositorio.BuscarPorId(tipoEvento.Id);

    if (eventoASerAtualizado == null)
    {
        return NotFound();
    }

    try
    {
        TipoEventoRepositorio.Alterar(tipoEvento);
        return Ok();
    }
    catch
    {
        return BadRequest();
    }
}
```

Alterar o repositório para deletar um registro

```
public void Deletar(int id)
{
    using (SqlConnection con = new SqlConnection(stringDeConexao))
    {
        string comandoSQL = "DELETE FROM TIPOS_EVENTOS WHERE ID = @ID;";
        SqlCommand cmd = new SqlCommand(comandoSQL, con);
        cmd.Parameters.AddWithValue("@ID", id);
        con.Open();
        cmd.ExecuteNonQuery();
    }
}
```

Deletar um tipo de evento

```
/// <summary>
/// Deleta um tipo de evento dado um determinado id
/// </summary>
/// <param name="id"></param>
/// <returns>Retorna a lista atualizada</returns>
[HttpDelete("{id}")]
public void Delete(int id)
{
    //eventos.Remove(eventos.Find(x => x.Id == id));
    //return Ok(eventos);
    TipoEventoRepositorio.Deletar(id);
}
```

InstituicoesController

```
[Route("api/[controller]")]
[ApiController]
public class InstituicoesController : ControllerBase
{

    // repositório

    [HttpGet]
    public IActionResult Get()
    {
        return Ok();
    }

    [HttpGet("{id}")]
    public IActionResult GetById(int id)
    {
        return Ok();
    }

    [HttpPost]
    public IActionResult Post()
    {
        return Ok();
    }

    [HttpPut]
    public IActionResult Put()
    {
        return Ok();
    }
}
```

Adicionar um novo domínio

```
public class InstituicaoDomain
{
    public int Id { get; set; }
    public string NomeFantasia { get; set; }
    public string RazaoSocial { get; set; }
    public string CNPJ { get; set; }
    public string Logradouro { get; set; }
    public string Cep { get; set; }
    public string Uf { get; set; }
    public string Cidade { get; set; }
}
```

Olhar todas as propriedades do banco de dados.

Criar interface

```
public interface IInstituicaoRepository
{
    void Cadastrar(InstituicaoDomain instituicao);
    void Alterar(InstituicaoDomain instituicao);
    InstituicaoDomain BuscarPorId(int id);
    List<InstituicaoDomain> Listar();
}
```

Criar repositório

```
public class InstituicaoRepository : IInstituicaoRepository
{
    public void Alterar(InstituicaoDomain instituicao)
    {
        throw new NotImplementedException();
    }

    public InstituicaoDomain BuscarPorId(int id)
    {
        throw new NotImplementedException();
    }

    public void Cadastrar(InstituicaoDomain instituicao)
    {
        throw new NotImplementedException();
    }

    public List<InstituicaoDomain> Listar()
    {
        throw new NotImplementedException();
    }
}
```

Ajustar os controllers

```
[Route("api/[controller]")]
[ApiController]
public class InstituicoesController : ControllerBase
{
    private IInstituicaoRepository InstituicaoRepository { get; set; }
    // repositório
    public InstituicoesController()
    {
        InstituicaoRepository = new InstituicaoRepository();
    }

    [HttpGet]
    public IActionResult Get()
    {
        return Ok(InstituicaoRepository.Listar());
    }

    [HttpGet("{id}")]
    public IActionResult GetById(int id)
    {
        InstituicaoDomain instituicao = InstituicaoRepository.BuscarPorId(id);
        if (instituicao == null)
        {
            return NotFound();
        }
        return Ok(instituicao);
    }

    [HttpPost]
    public IActionResult Post(InstituicaoDomain instituicao)
    {
        try
        {
            InstituicaoRepository.Cadastrar(instituicao);
            return Ok();
        }
        catch
        {
            return BadRequest();
        }

    }

    [HttpPut("{id}")]
    public IActionResult Put(int id, InstituicaoDomain instituicao)
    {
        InstituicaoDomain buscada = InstituicaoRepository.BuscarPorId(id);
        if (buscada == null)
        {
            return NotFound();
        }
        try
        {
            instituicao.Id = id;
            InstituicaoRepository.Cadastrar(instituicao);
            return Ok();
        }
        catch
        {
            return BadRequest();
        }

    }
}
```

Ajustar os repositórios

```
public List<InstituicaoDomain> Listar()
{
    string queryASerExecutada = "SELECT ID, NOME_FANTASIA, RAZAO_SOCIAL, CNPJ, CEP, LOGRADOURO, UF, CIDADE FROM INSTITUICOES";

    // crio uma nova lista
    List<InstituicaoDomain> listaInstituicoes = new List<InstituicaoDomain>();

    // garante que os recursos sejam fechados e descartados quando o código será encerrado
    // sqlconnection cria uma instância do objeto
    using (SqlConnection con = new SqlConnection(stringDeConexao))
    {
        // permite que você crie queries e envie para o banco de dados
        using (SqlCommand cmd = new SqlCommand(queryASerExecutada, con))
        {
            // para determinar que é do tipo texto
            // cmd.CommandType = CommandType.Text;
            con.Open();
            // retorna um conjunto de resultados do banco de dados
            SqlDataReader rdr = cmd.ExecuteReader();
            // enquanto tiver registros
            while (rdr.Read())
            {
                // cria um novo objeto e insere na lista
                InstituicaoDomain instituicao = new InstituicaoDomain
                {
                    Id = Convert.ToInt32(rdr["ID"]),
                    NomeFantasia = rdr["NOME_FANTASIA"].ToString(),
                    RazaoSocial = rdr["RAZAO_SOCIAL"].ToString(),
                    CNPJ = rdr["CNPJ"].ToString(),
                    Cep = rdr["CEP"].ToString(),
                    Logradouro = rdr["LOGRADOURO"].ToString(),
                    Uf = rdr["UF"].ToString(),
                    Cidade = rdr["CIDADE"].ToString(),
                };

                // adiciona cada evento na lista
                listaInstituicoes.Add(instituicao);
            }
        }
    }
    // retorna a própria lista
    return listaInstituicoes;
}
```

```
public InstituicaoDomain BuscarPorId(int id)
{
    InstituicaoDomain instituicao = new InstituicaoDomain();
    using (SqlConnection con = new SqlConnection(stringDeConexao))
    {
        string comandoSQL = "SELECT * FROM INSTITUICOES WHERE ID = @ID ORDER BY ID ASC";
        SqlCommand cmd = new SqlCommand(comandoSQL, con);
        cmd.Parameters.AddWithValue("@ID", id);
        con.Open();
        SqlDataReader lerOsRegistros = cmd.ExecuteReader();
        if (lerOsRegistros.HasRows)
        {
            while (lerOsRegistros.Read())
            {
                instituicao.Id = Int32.Parse(lerOsRegistros["ID"].ToString());
                instituicao.RazaoSocial = lerOsRegistros["RAZAO_SOCIAL"].ToString();
                instituicao.NomeFantasia = lerOsRegistros["NOME_FANTASIA"].ToString();
                instituicao.CNPJ = lerOsRegistros["CNPJ"].ToString();
                instituicao.Cep = lerOsRegistros["CEP"].ToString();
                instituicao.Logradouro = lerOsRegistros["LOGRADOURO"].ToString();
                instituicao.Uf = lerOsRegistros["UF"].ToString();
                instituicao.Cidade = lerOsRegistros["CIDADE"].ToString();
            }
            return instituicao;
        }
    }

    return null;
}
```

```
public void Cadastrar(InstituicaoDomain instituicao)
{
    using (SqlConnection con = new SqlConnection(stringDeConexao))
    {
        string comandoSQL = "INSERT INTO INSTITUICOES (RAZAO_SOCIAL, NOME_FANTASIA, CNPJ, LOGRADOURO, CEP, UF, CIDADE) VALUES( @RAZAO_SOCIAL, @NOME_FANTASIA, @CNPJ, @LOGRADOURO, @CEP, @UF, @CIDADE)";
        SqlCommand cmd = new SqlCommand(comandoSQL, con);
        cmd.Parameters.AddWithValue("@RAZAO_SOCIAL", instituicao.RazaoSocial);
        cmd.Parameters.AddWithValue("@NOME_FANTASIA", instituicao.NomeFantasia);
        cmd.Parameters.AddWithValue("@CNPJ", instituicao.CNPJ);
        cmd.Parameters.AddWithValue("@LOGRADOURO", instituicao.Logradouro);
        cmd.Parameters.AddWithValue("@CEP", instituicao.Cep);
        cmd.Parameters.AddWithValue("@UF", instituicao.Uf);
        cmd.Parameters.AddWithValue("@CIDADE", instituicao.Cidade);
        con.Open();
        cmd.ExecuteNonQuery();
    }
}
```

```
public void Alterar(InstituicaoDomain instituicao)
{
    using (SqlConnection con = new SqlConnection(stringDeConexao))
    {
        string comandoSQL = "UPDATE INSTITUICOES SET RAZAO_SOCIAL = @RAZAO_SOCIAL" +
            ", NOME_FANTASIA = @NOME_FANTASIA " +
            ", CNPJ = @CNPJ " +
            ", LOGRADOURO = @LOGRADOURO " +
            ", CEP = @CEP " +
            ", UF = @UF " +
            ", CIDADE = @CIDADE " +
            " WHERE ID = @ID";

        SqlCommand cmd = new SqlCommand(comandoSQL, con);

        cmd.Parameters.AddWithValue("@ID", instituicao.Id);
        cmd.Parameters.AddWithValue("@RAZAO_SOCIAL", instituicao.RazaoSocial);
        cmd.Parameters.AddWithValue("@NOME_FANTASIA", instituicao.NomeFantasia);
        cmd.Parameters.AddWithValue("@CNPJ", instituicao.CNPJ);
        cmd.Parameters.AddWithValue("@LOGRADOURO", instituicao.Logradouro);
        cmd.Parameters.AddWithValue("@CEP", instituicao.Cep);
        cmd.Parameters.AddWithValue("@UF", instituicao.Uf);
        cmd.Parameters.AddWithValue("@CIDADE", instituicao.Cidade);

        con.Open();
        cmd.ExecuteNonQuery();
    }

}
```

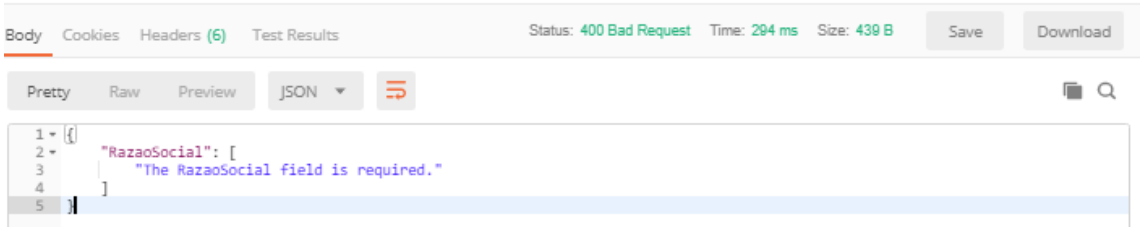
ENDPOINTS

DOCUMENTAÇÃO

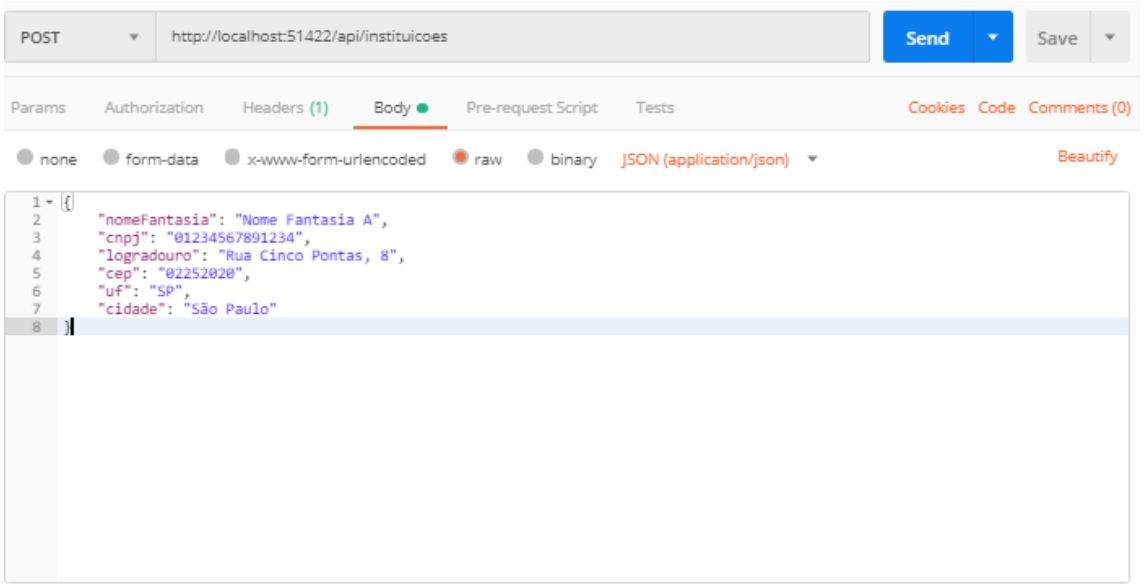
https://docs.microsoft.com/pt-br/aspnet/core/tutorials/getting-started-with-swashbuckle?view=aspnetcore-2.2&tabs=visual-studio

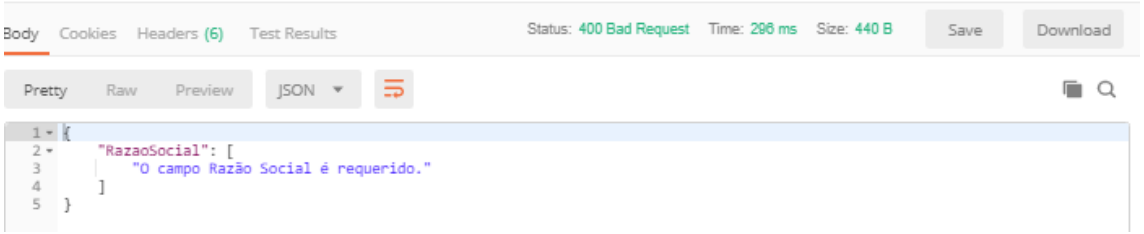
VALIDAR DADOS DE ENTRADA COM ANOTAÇÕES

```
public class InstituicaoDomain
{
    4 referências | 0 exceções
    public int Id { get; set; }
    4 referências | 0 exceções
    public string NomeFantasia { get; set; }
    [Required]
    4 referências | 0 exceções
    public string RazaoSocial { get; set; }
    4 referências | 0 exceções
    public string CNPJ { get; set; }
    4 referências | 0 exceções
    public string Logradouro { get; set; }
    4 referências | 0 exceções
    public string Cep { get; set; }
    4 referências | 0 exceções
    public string Uf { get; set; }
    4 referências | 0 exceções
    public string Cidade { get; set; }
}
```




```
18 referências
public class InstituicaoDomain
{
    4 referências | 0 exceções
    public int Id { get; set; }
    4 referências | 0 exceções
    public string NomeFantasia { get; set; }
    [Required(ErrorMessage = "O campo Razão Social é requerido.")]
    4 referências | 0 exceções
    public string RazaoSocial { get; set; }
    4 referências | 0 exceções
    public string CNPJ { get; set; }
    4 referências | 0 exceções
    public string Logradouro { get; set; }
    4 referências | 0 exceções
    public string Cep { get; set; }
    4 referências | 0 exceções
    public string Uf { get; set; }
    4 referências | 0 exceções
    public string Cidade { get; set; }
}
```






```
public class InstituicaoDomain
{
    4 referências | 0 exceções
    public int Id { get; set; }
    4 referências | 0 exceções
    public string NomeFantasia { get; set; }
    [Required(ErrorMessage = "O campo Razão Social é requerido.")]
    4 referências | 0 exceções
    public string RazaoSocial { get; set; }
    4 referências | 0 exceções
    public string CNPJ { get; set; }
    4 referências | 0 exceções
    public string Logradouro { get; set; }
    4 referências | 0 exceções
    public string Cep { get; set; }
    [StringLength(2, MinimumLength = 2)]
    4 referências | 0 exceções
    public string Uf { get; set; }
    4 referências | 0 exceções
    public string Cidade { get; set; }
}
```

```
{
    "Uf": [
        "The field Uf must be a string with a minimum length of 2 and a maximum length of 2."
    ],
    "RazaoSocial": [
        "O campo Razão Social é requerido."
    ]
}
```

```
public class InstituicaoDomain
{
    4 referências | 0 exceções
    public int Id { get; set; }
    4 referências | 0 exceções
    public string NomeFantasia { get; set; }
    [Required(ErrorMessage = "O campo Razão Social é requerido.")]
    4 referências | 0 exceções
    public string RazaoSocial { get; set; }
    4 referências | 0 exceções
    public string CNPJ { get; set; }
    4 referências | 0 exceções
    public string Logradouro { get; set; }
    4 referências | 0 exceções
    public string Cep { get; set; }
    [StringLength(2, MinimumLength = 2, ErrorMessage = "O campo de UF deve conter exatamente dois caracteres.")]
    4 referências | 0 exceções
    public string Uf { get; set; }
    4 referências | 0 exceções
    public string Cidade { get; set; }
}
```


```
{
    "Uf": [
        "O campo de UF deve conter exatamente dois caracteres."
    ],
    "RazaoSocial": [
        "O campo Razão Social é requerido."
    ]
}
```

Criando um novo UsuarioDomain

```
0 referências
public class UsuarioDomain
{
    0 referências | 0 exceções
    public int Id { get; set; }
    0 referências | 0 exceções
    public string Nome { get; set; }
    0 referências | 0 exceções
    public string Email { get; set; }
    0 referências | 0 exceções
    public string Senha { get; set; }
    0 referências | 0 exceções
    public string TipoUsuario { get; set; }

}
```

```
0 referências
public class UsuarioDomain
{
    0 referências | 0 exceções
    public int Id { get; set; }
    0 referências | 0 exceções
    public string Nome { get; set; }
    [Required]
    0 referências | 0 exceções
    public string Email { get; set; }
    [Required]
    0 referências | 0 exceções
    public string Senha { get; set; }
    0 referências | 0 exceções
    public string TipoUsuario { get; set; }

}
```

UsuariosController

Vamos apenas cadastrar um novo usuário para depois fazermos o login

```
namespace Senai.Svigufo.Api.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    0 referências
    public class UsuariosController : Controller
    {

        [HttpPost]
        0 referências | 0 solicitações | 0 exceções
        public IActionResult Post(UsuarioDomain usuario)
        {
            return Ok();
        }

    }
}
```

Criar a Interface para cadastrar

```
1 referência
public interface IUsuarioRepository
{
    1 referência | 0 exceções
    void Cadastrar(UsuarioDomain usuario);
}
```

Criar o repositório para acesso ao banco de dados

```
0 referências
public class UsuarioRepository : IUsuarioRepository
{
    1 referência | 0 exceções
    public void Cadastrar(UsuarioDomain usuario)
    {
        throw new NotImplementedException();
    }
}
```

Incluir a implementação no repositório

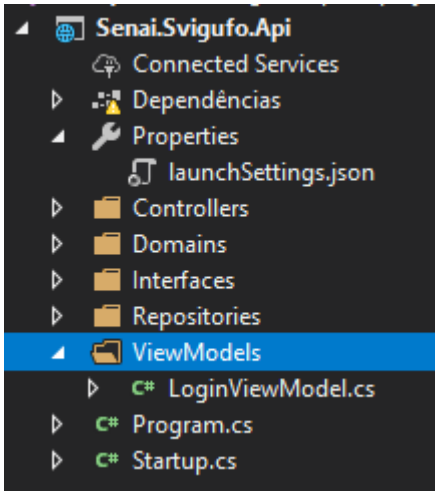
```
1 referência | 0 exceções
public void Cadastrar(UsuarioDomain usuario)
{
    string QueryInsert = "INSERT INTO USUARIOS (NOME, EMAIL, SENHA, TIPO_USUARIO) VALUES (@NOME, @EMAIL, @SENHA, @TIPO_USUARIO)";

    using (SqlConnection con = new SqlConnection(stringDeConexao))
    {

        con.Open();

        using (SqlCommand cmd = new SqlCommand(QueryInsert, con))
        {
            cmd.Parameters.AddWithValue("@NOME", usuario.Nome);
            cmd.Parameters.AddWithValue("@EMAIL", usuario.Email);
            cmd.Parameters.AddWithValue("@SENHA", usuario.Senha);
            cmd.Parameters.AddWithValue("@TIPO_USUARIO", usuario.TipoUsuario);

            cmd.ExecuteNonQuery();
        }

    }
}
```

Incluir no construtor o repositório

```
public class UsuariosController : Controller
{

    1 referência | 0 exceções
    private IUsuarioRepository UsuarioRepository { get; set; }

    0 referências | 0 exceções
    public UsuariosController()
    {
        UsuarioRepository = new UsuarioRepository();
    }

    [HttpPost]
    0 referências | 0 solicitações | 0 exceções
    public IActionResult Post(UsuarioDomain usuario)
    {
        return Ok();
    }

}
```

```
[HttpPost]
0 referências | 0 solicitações | 0 exceções
public IActionResult Post(UsuarioDomain usuario)
{
    try
    {
        UsuarioRepository.Cadastrar(usuario);
        return Ok();
    }
    catch
    {
        return BadRequest();
    }

}
```

Posso criar um de visualização para receber apenas alguns dados de entrada



```
0 referências
public class LoginViewModel
{
    [Required]
    0 referências | 0 exceções
    public string Email { get; set; }
    0 referências | 0 exceções
    public string Password { get; set; }
}
```

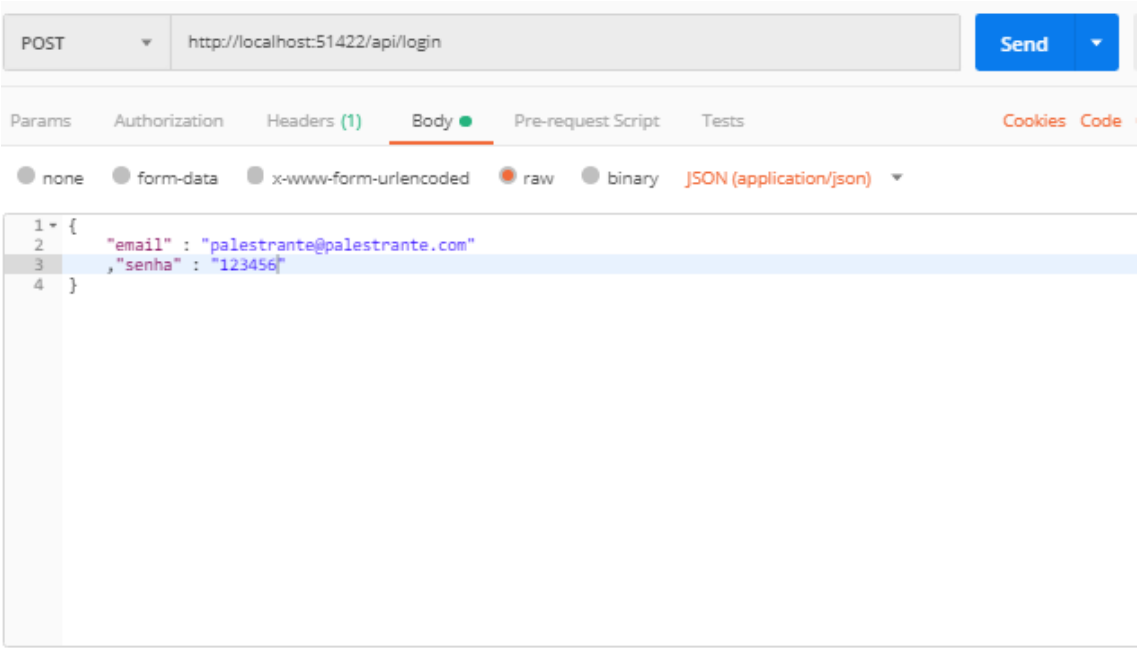
```
namespace Senai.Svigufo.Api.Controllers
{
    [Produces("application/json")]
    [Route("api/[controller]")]
    [ApiController]
    1 referência
    public class LoginController : ControllerBase
    {

        1 referência | 0 exceções
        private IUsuarioRepository UsuarioRepository { get; set; }

        0 referências | 0 exceções
        public LoginController()
        {
            UsuarioRepository = new UsuarioRepository();
        }


        /// <summary>
        /// Buscar um usuario por email e senha
        /// </summary>
        /// <param name="usuario"></param>
        /// <returns>Retorna uma autenticação para o usuário</returns>
        [HttpPost]
        0 referências | 0 solicitações | 0 exceções
        public IActionResult Post(LoginViewModel login)
        {
            return Ok();
        }

    }
}
```

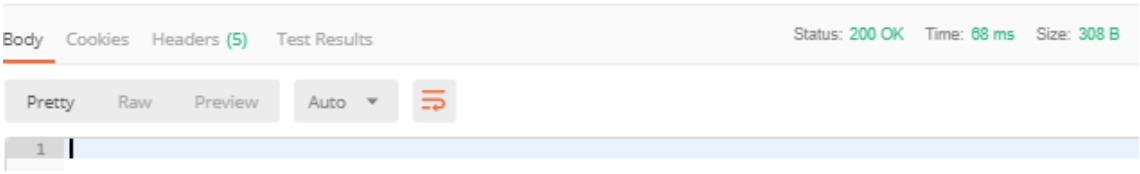
ALTERAR PARA RECEBER EMAIL E SENHA COMO STRING

```
2 referências
public class UsuarioRepository : IUsuarioRepository
{
    private string stringDeConexao = "Data Source=localhost;Initial Catalog=SENAI_SVIGUFO;Integrated Security=True";

    1 referência | 0 exceções
    public UsuarioDomain BuscarPorEmailESenha(LoginViewModel login)
    {
        throw new NotImplementedException();
    }

    2 referências | 0 exceções
    public void Cadastrar(UsuarioDomain usuario)...
}
```

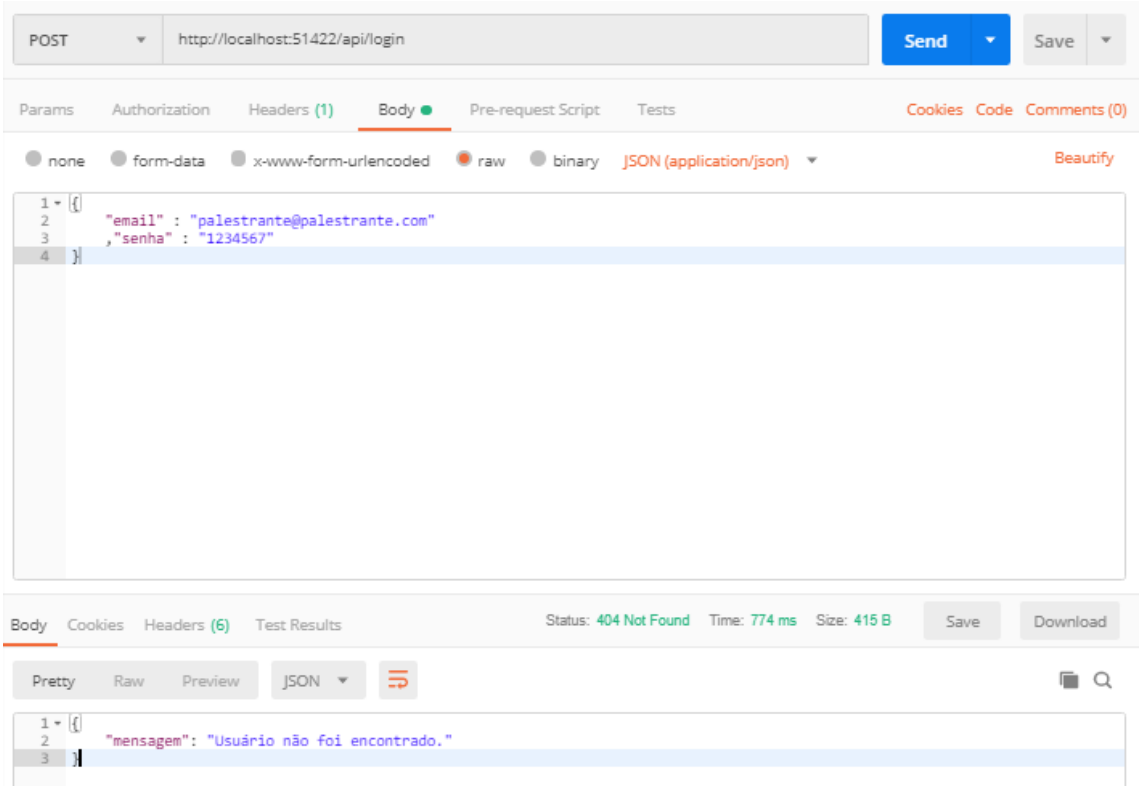
```
2 referências | 0 exceções
public UsuarioDomain BuscarPorEmailESenha(LoginViewModel login)
{
    string QueryEmailSenha = "SELECT ID, NOME, EMAIL, TIPO_USUARIO FROM USUARIOS WHERE EMAIL = @EMAIL AND SENHA = @SENHA";

    using (SqlConnection con = new SqlConnection(stringDeConexao))
    {
        using (SqlCommand cmd = new SqlCommand(QueryEmailSenha, con))
        {
            cmd.Parameters.AddWithValue("@EMAIL", login.Email);
            cmd.Parameters.AddWithValue("@SENHA", login.Senha);
            con.Open();

            SqlDataReader rdr = cmd.ExecuteReader();
            if (rdr.HasRows)
            {
                UsuarioDomain usuarioBuscado = new UsuarioDomain();
                while (rdr.Read())
                {
                    usuarioBuscado.Id = Int32.Parse(rdr["ID"].ToString());
                    usuarioBuscado.Email = rdr["EMAIL"].ToString();
                    usuarioBuscado.Nome = rdr["NOME"].ToString();
                    usuarioBuscado.TipoUsuario = rdr["TIPO_USUARIO"].ToString();
                }
                return usuarioBuscado;
            }

        }
        return null;
    }

}
```

```
/// <summary>
/// Buscar um usuario por email e senha
/// </summary>
/// <param name="usuario"></param>
/// <returns>Retorna uma autenticação para o usuário</returns>
[HttpPost]
0 referências | 0 solicitações | 0 exceções
public IActionResult Post(LoginViewModel login)
{
    try
    {
        UsuarioDomain usuario = UsuarioRepository.BuscarPorEmailESenha(login);
        if (usuario == null)
        {
            return NotFound("Usuário não foi encontrado.");
        }
        return Ok();
    }
    catch
    {
        return BadRequest();
    }

}
```






```
[HttpPost]
0 referências | 0 solicitações | 0 exceções
public IActionResult Post(LoginViewModel login)
{
    try
    {
        UsuarioDomain usuario = UsuarioRepository.BuscarPorEmailESenha(login);
        if (usuario == null)
        {
            return NotFound(
                new {
                    mensagem = "Usuário não foi encontrado."
                }
            );
        }
        return Ok();
    }
    catch
    {
        return BadRequest();
    }

}
```


POST
http://localhost:51422/api/login
Send
Save
Params
Authorization
Headers (1)
Body
Pre-request Script
Tests
Cookies
Code
Comments (0)
none
form-data
x-www-form-urlencoded
raw
binary
JSON (application/json)
Beautify
{
"email" : "palestrante@palestrante.com"
,"senha" : "1234567"
}
Body
Cookies
Headers (6)
Test Results
Status: 404 Not Found
Time: 774 ms
Size: 415 B
Save
Download
Pretty
Raw
Preview
JSON
{
"mensagem": "Usuário não foi encontrado."
}

Autenticação JWT

System.IdentityModel.Tokens.Jwt – criar e validar o jwt

Microsoft.AspNetCore.Authentication.JwtBearer - Integrar a parte de autenticação

Startup.cs

```
services.AddAuthentication(options =>
{
    // esquema de autenticação padrão
    // Bearer - quem será o portador do jwt
    options.DefaultAuthenticateScheme = "JwtBearer";
    options.DefaultChallengeScheme = "JwtBearer";
}).AddJwtBearer("JwtBearer", options =>
{
    // parâmetros para autenticar
    options.TokenValidationParameters = new Microsoft.IdentityModel.Tokens.TokenValidationParameters
    {
        // quando um token for enviado na requisição, preciso saber como será sua validação
        // quem está solicitando
        ValidateIssuer = true,
        // quem está realizando a audiência
        ValidateAudience = true,
        // tempo de expiração
        ValidateLifetime = true,
        // chave de assinatura que será utilizada por quem está validando
        IssuerSigningKey = new SymmetricSecurityKey(System.Text.Encoding.UTF8.GetBytes("svigufo-chave-autenticacao")),
        // tempo de expiração
        ClockSkew = TimeSpan.FromMinutes(5),
        // quem está fazendo a própria validação
        ValidIssuer = "Svigufo.WebApi",
        // de onde está vindo
        ValidAudience = "Svigufo.WebApi"
    };
});
```

```
0 referências | 0 exceções
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{

    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    app.UseAuthentication();

    app.UseSwagger();

    app.UseSwaggerUI(c =>
    {
        c.SwaggerEndpoint("/swagger/v1/swagger.json", "Svigufo API V1");
    });

    app.UseCors(builder => builder.AllowAnyHeader().AllowAnyMethod().AllowAnyOrigin());

    app.UseMvc();

}
```

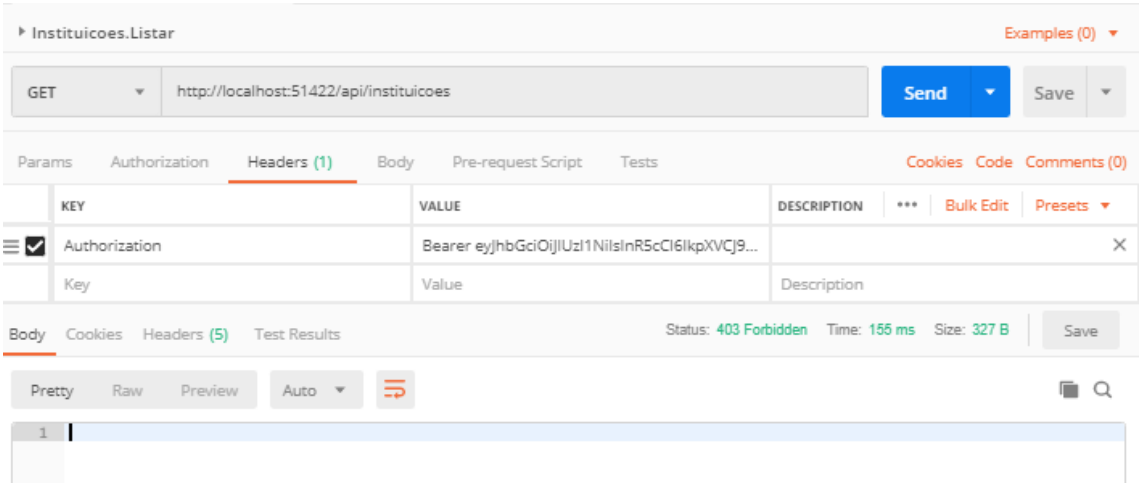
```
[HttpPost]
0 referências | 0 solicitações | 0 exceções
public IActionResult Post(LoginViewModel login)
{
    try
    {
        UsuarioDomain usuario = UsuarioRepository.BuscarPorEmailESenha(login);
        if (usuario == null)
        {
            return NotFound(
                new
                {
                    mensagem = "Usuário não foi encontrado."
                }
            );
        }
        var claims = new[]
        {
            new Claim(JwtRegisteredClaimNames.Email, login.Email),
            new Claim(JwtRegisteredClaimNames.Jti, usuario.Id.ToString()),
            new Claim(ClaimTypes.Role, usuario.TipoUsuario),
        };

        //recebe uma instancia da classe SymmetricSecurityKey
        //armazenando a chave de criptografia usada na criação do token
        var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes("svigufo-chave-autenticacao"));

        //recebe um objeto do tipo SigninCredentials contendo a chave de
        //criptografia e o algoritmo de segurança empregados na geração
        // de assinaturas digitais para tokens
        var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

        var token = new JwtSecurityToken(
             issuer: "Svigufo.WebApi",
             audience: "Svigufo.WebApi",
             claims: claims,
             expires: DateTime.Now.AddMinutes(30),
             signingCredentials: creds);

        return Ok(new
        {
            token = new JwtSecurityTokenHandler().WriteToken(token)
        });
    }
    catch
    {
        return BadRequest();
    }

}
```

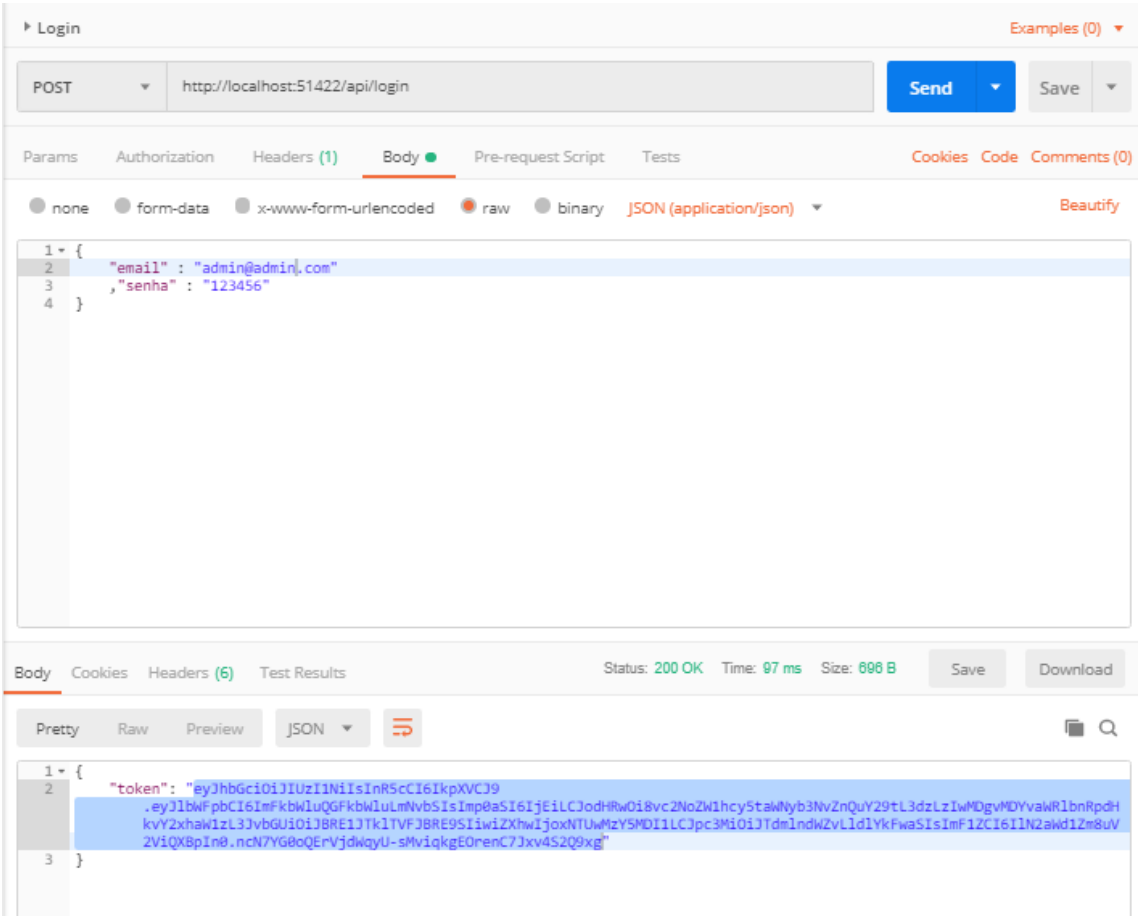
InstituicoesController

```
/// <summary>
/// Buscar a lista de instituições
/// </summary>
/// <returns>Lista de Instituições</returns>
[Authorize]
[HttpGet]
0 referências | 0 solicitações | 0 exceções
public IActionResult Get()
{
    return Ok(InstituicaoRepository.Listar());
}
```

Testando com regras de acesso e perfil

```
/// <summary>
/// Buscar a lista de instituições
/// </summary>
/// <returns>Lista de Instituições</returns>
[Authorize(Roles = "ADMINISTRADOR")]
[HttpGet]
0 referências | 0 solicitações | 0 exceções
public IActionResult Get()
{
    return Ok(InstituicaoRepository.Listar());
}
```





Ajustar todos os endpoints com o devido acesso

Criação dos Eventos

```
namespace Senai.Svigufo.Api.Domains
{
    /// <summary>
    /// Domínio do Evento
    /// </summary>
    10 referências
    public class EventoDomain
    {
        1 referência | 0 exceções
        public int Id { get; set; }
        1 referência | 0 exceções
        public string Titulo { get; set; }
        1 referência | 0 exceções
        public string Descricao { get; set; }
        1 referência | 0 exceções
        public DateTime DataEvento { get; set; }
        1 referência | 0 exceções
        public bool AcessoLivre { get; set; }
        0 referências | 0 exceções
        public TipoEventoDomain TipoEvento { get; set; }
        0 referências | 0 exceções
        public InstituicaoDomain Instituicao { get; set; }
    }
}
```

EventosController

```
namespace Senai.Svigufo.Api.Controllers
{
    [Produces("application/json")]
    [Route("api/[controller]")]
    [ApiController]
    0 referências
    public class EventosController : ControllerBase
    {

    }
}
```

IEventoRepository

```
namespace Senai.Svigufo.Api.Interfaces
{
    0 referências
    public interface IEventoRepository
    {
        /// <summary>
        /// Listar todos os eventos
        /// </summary>
        /// <returns>Retorna uma lista de eventos</returns>
        0 referências | 0 exceções
        List<EventoDomain> Listar();

        /// <summary>
        /// Cadastrar um novo evento
        /// </summary>
        /// <param name="evento">EventoDomain</param>
        0 referências | 0 exceções
        void Cadastrar(EventoDomain evento);

        /// <summary>
        /// Atualizar um evento existente
        /// </summary>
        /// <param name="id">Id</param>
        /// <param name="evento">EventoDomain</param>
        0 referências | 0 exceções
        void Atualizar(int id, EventoDomain evento);
    }
}
```

EventoRepository

```
namespace Senai.Svigufo.Api.Repositories
{
    0 referências
    public class EventoRepository : IEventoRepository
    {
        1 referência | 0 exceções
        public void Atualizar(int id, EventoDomain evento)
        {
            throw new System.NotImplementedException();
        }

        1 referência | 0 exceções
        public void Cadastrar(EventoDomain evento)
        {
            throw new System.NotImplementedException();
        }

        1 referência | 0 exceções
        public List<EventoDomain> Listar()
        {
            throw new System.NotImplementedException();
        }
    }
}
```

EventosController

```
[Produces("application/json")]
[Route("api/[controller]")]
[ApiController]
1 referência
public class EventosController : ControllerBase
{

    1 referência | 0 exceções
    public IEventoRepository EventoRepositorio { get; set; }

    0 referências | 0 exceções
    public EventosController()
    {
        EventoRepositorio = new EventoRepository();
    }

}
```

```
2 referências | 0 exceções
public List<EventoDomain> Listar()
{
    string queryASerExecutada = "SELECT ID, TITULO, DESCRICAO, DATA_EVENTO, ACESSO_LIVRE, ID_INSTITUICAO, ID_TIPO_EVENTO FROM EVENTOS";

    List<EventoDomain> eventos = new List<EventoDomain>();

    using (SqlConnection con = new SqlConnection(stringDeConexao))
    {

        using (SqlCommand cmd = new SqlCommand(queryASerExecutada, con))
        {

            con.Open();

            SqlDataReader rdr = cmd.ExecuteReader();

            while (rdr.Read())
            {

                EventoDomain evento = new EventoDomain
                {
                    Id = Convert.ToInt32(rdr["ID"]),
                    Titulo = rdr["TITULO"].ToString(),
                    Descricao = rdr["DESCRICAO"].ToString(),
                    DataEvento = Convert.ToDateTime(rdr["DATA_EVENTO"].ToString()),
                    AcessoLivre = rdr["ACESSO_LIVRE"].ToString().Equals("True") ? true : false,
                };

                eventos.Add(evento);
            }
        }
    }

    return eventos;
}
```

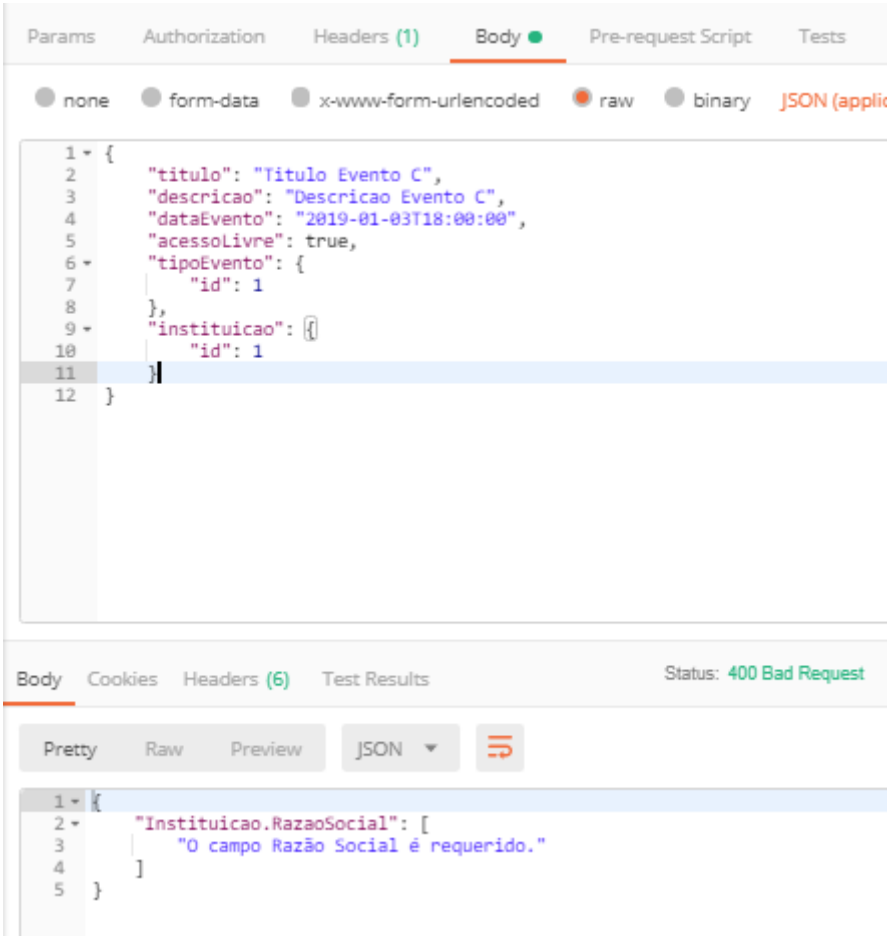
```
// string queryASerExecutada = "SELECT ID, TITULO, DESCRICAO, DATA_EVENTO, ACESSO_LIVRE, ID_INSTITUICAO, ID_TIPO_EVENTO FROM EVENTOS";
string queryASerExecutada = "SELECT E.ID AS ID_EVENTO, E.TITULO AS TITULO_EVENTO, E.DESCRICAO, E.DATA_EVENTO, E.ACESSO_LIVRE" +
    ", I.ID AS ID_INSTITUICAO, I.NOME_FANTASIA AS NOME_INSTITUICAO" +
    ", TE.ID AS ID_TIPO_EVENTO ,TE.TITULO AS TIPO_EVENTO " +
    "FROM EVENTOS E INNER JOIN INSTITUICOES I ON E.ID_INSTITUICAO = I.ID " +
    "INNER JOIN TIPOS_EVENTOS TE ON E.ID_TIPO_EVENTO = TE.ID;";
```

```
EventoDomain evento = new EventoDomain
{
    Id = Convert.ToInt32(rdr["ID_EVENTO"]),
    Titulo = rdr["TITULO_EVENTO"].ToString(),
    Descricao = rdr["DESCRICAO"].ToString(),
    DataEvento = Convert.ToDateTime(rdr["DATA_EVENTO"].ToString()),
    AcessoLivre = rdr["ACESSO_LIVRE"].ToString().Equals("True") ? true : false,
    Instituicao = new InstituicaoDomain
    {
        Id = Convert.ToInt32(rdr["ID_INSTITUICAO"]),
        NomeFantasia = rdr["NOME_INSTITUICAO"].ToString()
    },
    TipoEvento = new TipoEventoDomain
    {
        Id = Convert.ToInt32(rdr["ID_TIPO_EVENTO"]),
        Nome = rdr["TIPO_EVENTO"].ToString()
    }
};
```

IGNORAR OS NULOS NA SAÍDA DO JSON

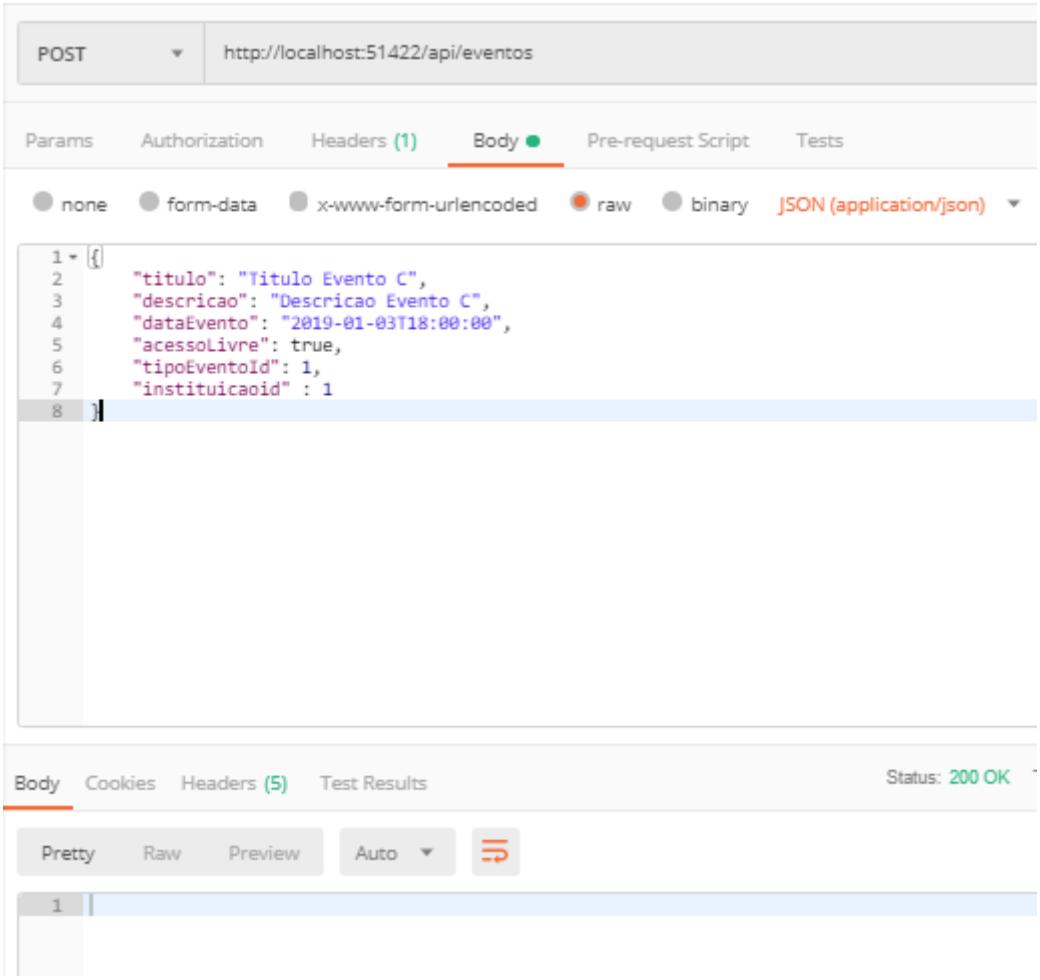
Startup.cs

```
public void ConfigureServices(IServiceCollection services)
{
     services.AddMvc()
             .AddJsonOptions(options => {
                options.SerializerSettings.NullValueHandling = NullValueHandling.Ignore;
     });
}
```




```
/// <summary>
/// Domínio do Evento
/// </summary>
11 referências
public class EventoDomain
{
    1 referência | 0 exceções
    public int Id { get; set; }
    2 referências | 0 exceções
    public string Titulo { get; set; }
    2 referências | 0 exceções
    public string Descricao { get; set; }
    2 referências | 0 exceções
    public DateTime DataEvento { get; set; }
    2 referências | 0 exceções
    public bool AcessoLivre { get; set; }
    0 referências | 0 exceções
    public int InstituicaoId { get; set; }
    0 referências | 0 exceções
    public int TipoEventoId { get; set; }
    2 referências | 0 exceções
    public TipoEventoDomain TipoEvento { get; set; }
    2 referências | 0 exceções
    public InstituicaoDomain Instituicao { get; set; }
}
```

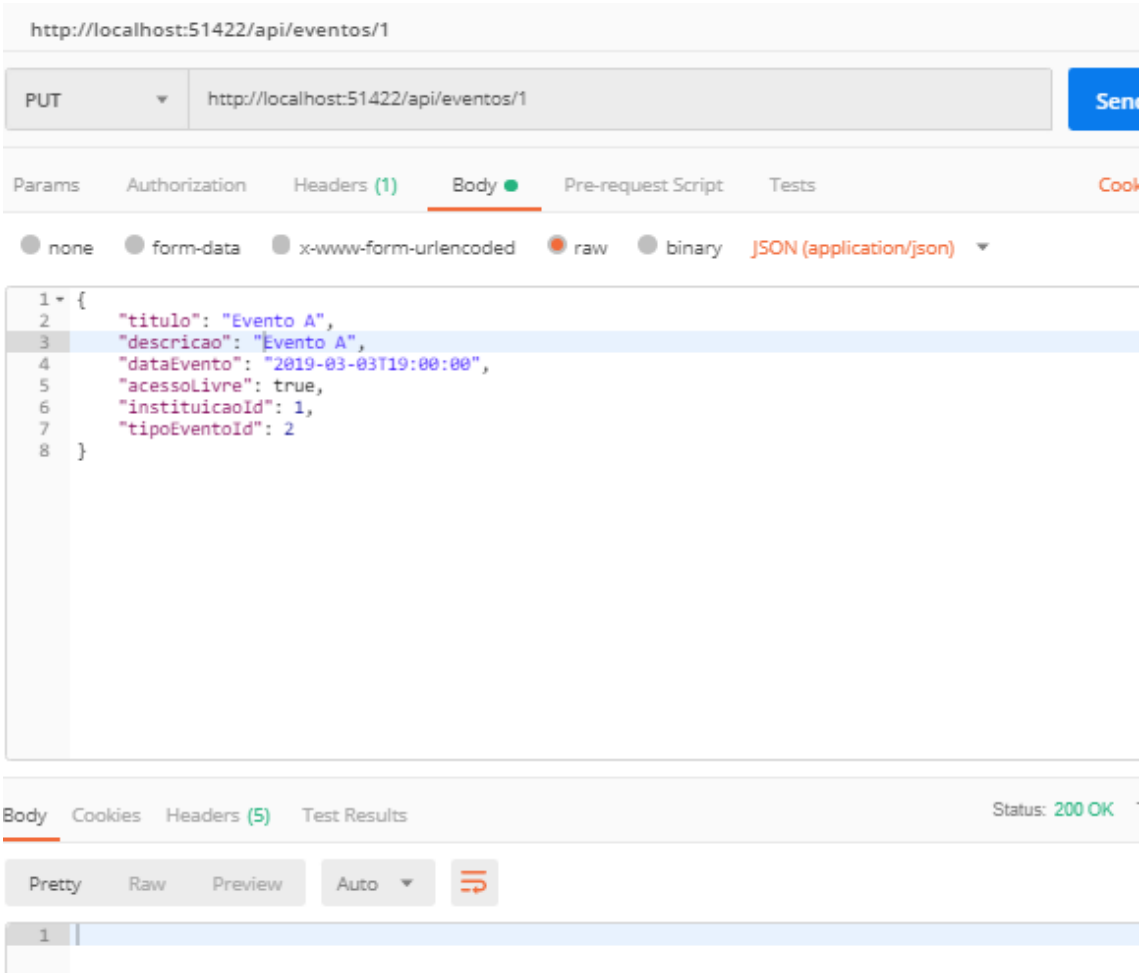
```
2 referências | 0 exceções
public void Cadastrar(EventoDomain evento)
{
    using (SqlConnection con = new SqlConnection(stringDeConexao))
    {
        string comandoSQL = "INSERT INTO EVENTOS (TITULO, DESCRICAO, DATA_EVENTO, ACESSO_LIVRE, ID_TIPO_EVENTO, ID_INSTITUICAO) " +
            "Values(@TITULO, @DESCRICAO, CONVERT(DATETIME, @DATA_EVENTO,120), @ACESSO_LIVRE, @ID_TIPO_EVENTO, @ID_INSTITUICAO)";
        SqlCommand cmd = new SqlCommand(comandoSQL, con);
        cmd.Parameters.AddWithValue("@TITULO", evento.Titulo);
        cmd.Parameters.AddWithValue("@DESCRICAO", evento.Descricao);
        cmd.Parameters.AddWithValue("@DATA_EVENTO", evento.DataEvento);
        cmd.Parameters.AddWithValue("@ACESSO_LIVRE", evento.AcessoLivre);
        cmd.Parameters.AddWithValue("@ID_TIPO_EVENTO", evento.TipoEventoId);
        cmd.Parameters.AddWithValue("@ID_INSTITUICAO", evento.InstituicaoId);
        con.Open();
        cmd.ExecuteNonQuery();
        con.Close();
    }
}
```




```
[HttpPut("{id}")]
0 referências | 0 solicitações | 0 exceções
public IActionResult Put(int id, EventoDomain evento)
{
    try
    {
        EventoRepositorio.Atualizar(id, evento);
        return Ok();
    }
    catch
    {
        return BadRequest();
    }
}
```

```
[HttpPut("{id}")]
0 referências | 0 solicitações | 0 exceções
public IActionResult Put(int id, EventoDomain evento)
{
    try
    {
        EventoRepositorio.Atualizar(id, evento);
        return Ok();
    }
    catch
    {
        return BadRequest();
    }
}
```

```
2 referências | 0 exceções
public void Atualizar(int id, EventoDomain evento)
{
    using (SqlConnection con = new SqlConnection(stringDeConexao))
    {
        string comandoSQL = "UPDATE EVENTOS SET " +
            "TITULO = @TITULO, " +
            "ID_INSTITUICAO = @ID_INSTITUICAO," +
            "DESCRICAO = @DESCRICAO, " +
            "DATA_EVENTO = CONVERT(DATETIME, @DATA_EVENTO, 120), " +
            "ACESSO_LIVRE = @ACESSO_LIVRE, " +
            "ID_TIPO_EVENTO = @ID_TIPO_EVENTO " +
            "WHERE ID = @ID";
        SqlCommand cmd = new SqlCommand(comandoSQL, con);
        cmd.Parameters.AddWithValue("@TITULO", evento.Titulo);
        cmd.Parameters.AddWithValue("@DESCRICAO", evento.Descricao);
        cmd.Parameters.AddWithValue("@DATA_EVENTO", evento.DataEvento);
        cmd.Parameters.AddWithValue("@ACESSO_LIVRE", evento.AcessoLivre);
        cmd.Parameters.AddWithValue("@ID_TIPO_EVENTO", evento.TipoEventoId);
        cmd.Parameters.AddWithValue("@ID_INSTITUICAO", evento.InstituicaoId);
        cmd.Parameters.AddWithValue("@ID", id);
        con.Open();
        cmd.ExecuteNonQuery();
        con.Close();
    }
}
```


http://localhost:51422/api/eventos/1
PUT
http://localhost:51422/api/eventos/1
Sen
Params
Authorization
Headers (1)
Body
Pre-request Script
Tests
Cook
none
form-data
x-www-form-urlencoded
raw
binary
JSON (application/json)
{
"titulo": "Evento A",
"descricao": "Evento A",
"dataEvento": "2019-03-03T19:00:00",
"acessoLivre": true,
"instituicaoId": 1,
"tipoEventoId": 2
}
Body
Cookies
Headers (5)
Test Results
Status: 200 OK
Pretty
Raw
Preview
Auto

ConviteDomain

```
public class ConviteDomain
{
    public int Id { get; set; }
    public int EventoId { get; set; }
    public EventoDomain Evento { get; set; }

    public int UsuarioId { get; set; }
    public UsuarioDomain Usuario { get; set; }

    public EnSituacaoConvite Situacao { get; set; }
}
```

```
public enum EnSituacaoConvite
{
    AGUARDANDO = 1,
    APROVADO = 2,
    //CONFIRMADO = 3,
    REPROVADO = 4,
    //CANCELADO = 5
}
```

INTERFACE CONVITE

```
public interface IConviteRepository
{
    /// <summary>
    /// Listar apenas os meus convites
    /// </summary>
    /// <returns>Lista dos meus convites</returns>
    List<ConviteDomain> ListarMeusConvites(int id);

    /// <summary>
    /// Listar todos os convites da plataforma
    /// </summary>
    /// <returns>Listar todos os convites</returns>
    List<ConviteDomain> Listar();

    /// <summary>
    /// Cadastra um novo convite
    /// </summary>
    /// <param name="convite">ConviteDomain</param>
    void Cadastrar(ConviteDomain convite);

    /// <summary>
    /// Altera um convite
    /// </summary>
    /// <param name="convite">ConviteDomain</param>
    void Alterar(ConviteDomain convite, int id);
```

Repositorio Convite Cadastrar

```
public void Cadastrar(ConviteDomain convite)
{
    string QueryInsert = @"INSERT INTO CONVITES(ID_EVENTO, ID_USUARIO, SITUACAO)
                           VALUES(@ID_EVENTO, @ID_USUARIO, @SITUACAO)";

    using(SqlConnection con = new SqlConnection(stringDeConexao))
    {
        con.Open();

        using(SqlCommand cmd = new SqlCommand(QueryInsert, con))
        {
            cmd.Parameters.AddWithValue("@ID_EVENTO", convite.EventoId);
            cmd.Parameters.AddWithValue("@ID_USUARIO", convite.UsuarioId);
            cmd.Parameters.AddWithValue("@SITUACAO", convite.Situacao);

            cmd.ExecuteNonQuery();
        }
    }
}
```

Controller Convite Post – Rota Convidar

```
[HttpPost]
[Authorize]
[Route("convidar")]
public IActionResult Convite(ConviteDomain convite)
{
    try
    {
        ConviteRepositorio.Cadastrar(convite);
        return Ok();
    }
    catch (Exception ex)
    {

        return BadRequest();
    }

}
```

Controller Convite Post – Rota Entrar

```
[Authorize]
[HttpPost("entrar/{eventoid}")]
public IActionResult Incricao(int eventoid)
{
    try
    {
        int usuarioid = Convert.ToInt32(HttpContext.User.Claims.First(c => c.Type == JwtRegisteredClaimNames.Jti).Value);

        ConviteDomain convite = new ConviteDomain
        {
            EventoId = eventoid,
            UsuarioId = usuarioid,
            Situacao = EnSituacaoConvite.AGUARDANDO
        };

        ConviteRepositorio.Cadastrar(convite);

        return Ok();
    }
    catch (Exception ex)
    {

        return BadRequest();
    }

}
```

Repositório Convite Alterar

```
public void Alterar(ConviteDomain convite, int id)
{
    string QueryUpdate = @"UPDATE CONVITES SET ID_EVENTO = @ID_EVENTO, ID_USUARIO = @ID_USUARIO, SITUACAO =
      @SITUACAO WHERE ID = @ID";

    using (SqlConnection con = new SqlConnection(stringDeConexao))
    {
        con.Open();

        using (SqlCommand cmd = new SqlCommand(QueryUpdate, con))
        {
            cmd.Parameters.AddWithValue("@ID", id);
            cmd.Parameters.AddWithValue("@ID_EVENTO", convite.EventoId);
            cmd.Parameters.AddWithValue("@ID_USUARIO", convite.UsuarioId);
            cmd.Parameters.AddWithValue("@SITUACAO", convite.Situacao);

            cmd.ExecuteNonQuery();
        }
    }
}
```

Controller Convite Alterar

```
[Authorize("ADMINISTRADOR")]
[HttpPut("{id}")]
public IActionResult Put(ConviteDomain convite, int id)
{
    try
    {
        ConviteRepositorio.Alterar(convite, id);
        return Ok();
    }
    catch (Exception ex)
    {
        return BadRequest();
    }
}
```

Repositorio Convite ListarTodos

```
public List<ConviteDomain> Listar()
{
    List<ConviteDomain> convites = new List<ConviteDomain>();
    string QuerySelect = @"SELECT C.ID AS ID_CONVITE,C.SITUACAO, E.ID AS ID_EVENTO,E.TITULO AS TITULO_EVENTO,E.DATA_EVENTO,E.ACESSO_LIVRE,TE.TITULO AS TIPO_EVENTO
                           ,U.ID AS ID_USUARIO, U.Email, U.Nome FROM CONVITES C INNER JOIN USUARIOS U ON C.ID_USUARIO = U.ID INNER JOIN EVENTOS E ON C.ID_EVENTO = E.ID
                    INNER JOIN TIPOS_EVENTOS TE ON E.ID_TIPO_EVENTO = TE.ID";

    using (SqlConnection con = new SqlConnection(stringDeConexao))
    {
        using (SqlCommand cmd = new SqlCommand(QuerySelect, con))
        {
            con.Open();
            SqlDataReader rdr = cmd.ExecuteReader();
            while (rdr.Read())
            {
                ConviteDomain convite = new ConviteDomain
                {
                    Id = Convert.ToInt32(rdr["ID_CONVITE"]),
                    Situacao = (EnSituacaoConvite)Convert.ToInt32(rdr["SITUACAO"]),
                    Evento = new EventoDomain
                    {
                        Titulo = rdr["TITULO_EVENTO"].ToString(),
                        DataEvento = Convert.ToDateTime(rdr["DATA_EVENTO"]),
                        AcessoLivre = Convert.ToBoolean(rdr["ACESSO_LIVRE"]),
                        TipoEvento = new TipoEventoDomain
                        {
                            Nome = rdr["TIPO_EVENTO"].ToString()
                        }
                    },
                    Usuario = new UsuarioDomain
                    {
                        Id = Convert.ToInt32(rdr["ID_USUARIO"]),
                        Nome = rdr["NOME"].ToString(),
                        Email = rdr["EMAIL"].ToString()
                    }
                };
                convites.Add(convite);
            }
        }
    }
```

Controller Convite Listar Todos – Administrador

```
[Authorize(Roles = "ADMINISTRADOR")]
[HttpGet]
[Route("listar")]
public IActionResult ListarTodos()
{
    return Ok(ConviteRepositorio.Listar());
}
```

Repositorio Convite Listar Meus Convites

```
public List<ConviteDomain> ListarMeusConvites(int id)
{

    List<ConviteDomain> convites = new List<ConviteDomain>();
    string QuerySelect = @"SELECT C.ID AS ID_CONVITE ,C.SITUACAO,E.ID AS ID_EVENTO,E.TITULO AS TITULO_EVENTO,E.DATA_EVENTO,E.ACESSO_LIVRE,TE.TITULO AS TIPO_EVENTO
                        FROM CONVITES C INNER JOIN EVENTOS E ON C.ID_EVENTO = E.ID INNER JOIN TIPOS_EVENTOS TE ON E.ID_TIPO_EVENTO = TE.ID WHERE C.ID_USUARIO = @ID;";

    using (SqlConnection con = new SqlConnection(stringDeConexao))
    {
        using (SqlCommand cmd = new SqlCommand(QuerySelect, con))
        {
            cmd.Parameters.AddWithValue("@ID", id);
            con.Open();

            SqlDataReader rdr = cmd.ExecuteReader();

            while (rdr.Read())
            {
                ConviteDomain convite = new ConviteDomain
                {
                    Id = Convert.ToInt32(rdr["ID_CONVITE"]),
                    Situacao = (EnSituacaoConvite) Convert.ToInt32(rdr["SITUACAO"]),
                    Evento = new EventoDomain
                    {
                        Titulo = rdr["TITULO_EVENTO"].ToString(),
                        DataEvento = Convert.ToDateTime(rdr["DATA_EVENTO"]),
                        TipoEvento = new TipoEventoDomain
                        {
                            Nome = rdr["TIPO_EVENTO"].ToString()
                        }
                    }
                };

                convites.Add(convite);
            }
        }
    }

    return convites;
}
```

Controller Convite Listar Meus Convites

```
[Authorize]
[HttpGet]
public IActionResult MeusConvites()
{
    int usuarioid = Convert.ToInt32(HttpContext.User.Claims.First(c => c.Type == JwtRegisteredClaimNames.Jti).Value);
    return Ok(ConviteRepositorio.ListarMeusConvites(usuarioid));
}
```

Ajustar Para que quando seja um evento privado o status seja Aguardando e quando seja publico o status seja Aprovado.

Criar na Interface IEventoRepository método EventoDomain BuscarPorId(int id)

Implementar no EventoRepositoy

```
public EventoDomain BuscarPorId(int id)
{
    // string queryASerExecutada = "SELECT ID, TITULO, DESCRICAO, DATA_EVENTO, ACESSO_LIVRE, ID_INSTITUICAO, ID_TIPO_EVENTO FROM EVENTOS";
    string queryASerExecutada = "SELECT E.ID AS ID_EVENTO, E.TITULO AS TITULO_EVENTO, E.DESCRICAO, E.DATA_EVENTO, E.ACESSO_LIVRE" +
        ", I.ID AS ID_INSTITUICAO, I.NOME_FANTASIA AS NOME_INSTITUICAO" +
        ", TE.ID AS ID_TIPO_EVENTO ,TE.TITULO AS TIPO_EVENTO " +
        "FROM EVENTOS E INNER JOIN INSTITUICOES I ON E.ID_INSTITUICAO = I.ID " +
        "INNER JOIN TIPOS_EVENTOS TE ON E.ID_TIPO_EVENTO = TE.ID" +
        "WHERE ID = @ID;";


    using (SqlConnection con = new SqlConnection(stringDeConexao))
    {

        using (SqlCommand cmd = new SqlCommand(queryASerExecutada, con))
        {
            cmd.Parameters.AddWithValue("@ID", id);
            con.Open();

            SqlDataReader rdr = cmd.ExecuteReader();

            if (rdr.HasRows)
            {
                while (rdr.Read())
                {
                    EventoDomain evento = new EventoDomain
                    {
                        Id = Convert.ToInt32(rdr["ID_EVENTO"]),
                        Titulo = rdr["TITULO_EVENTO"].ToString(),
                        Descricao = rdr["DESCRICAO"].ToString(),
                        DataEvento = Convert.ToDateTime(rdr["DATA_EVENTO"].ToString()),
                        AcessoLivre = rdr["ACESSO_LIVRE"].ToString().Equals("True") ? true : false,
                        Instituicao = new InstituicaoDomain
                        {
                            Id = Convert.ToInt32(rdr["ID_INSTITUICAO"]),
                            NomeFantasia = rdr["NOME_INSTITUICAO"].ToString()
                        },
                        TipoEvento = new TipoEventoDomain
                        {
                            Id = Convert.ToInt32(rdr["ID_TIPO_EVENTO"]),
                            Nome = rdr["TIPO_EVENTO"].ToString()
                        }
                    };

                    return evento;
                }
            }
        }
    }

    return null;
}
```

Ajustar no ConvitesController Inscricao e Convite

```
[Authorize]
[HttpPost("entrar/{eventoid}")]
0 references | corujasdevbr, 3 days ago | 1 author, 1 change
public IActionResult Incricao(int eventoid)
{
    try
    {
        int usuarioid = Convert.ToInt32(HttpContext.User.Claims.First(c => c.Type == JwtRegisteredClaimNames.Jti).Value);

        EventoDomain evento = EventoRepositorio.BuscarPorId(eventoid);

        if(evento != null)
        {
            return NotFound();
        }

        ConviteDomain convite = new ConviteDomain
        {
            EventoId = eventoid,
            UsuarioId = usuarioid,
            Situacao = (evento.AcessoLivre ? EnSituacaoConvite.APROVADO : EnSituacaoConvite.AGUARDANDO)
        };

        ConviteRepositorio.Cadastrar(convite);

        return Ok();
    }
    catch (Exception ex)
    {

        return BadRequest();
    }
}

[HttpPost]
[Authorize]
[Route("convidar")]
0 references | corujasdevbr, 3 days ago | 1 author, 1 change
public IActionResult Convite(ConviteDomain convite)
{
    try
    {
        EventoDomain evento = EventoRepositorio.BuscarPorId(convite.EventoId);

        if (evento != null)
        {
            return NotFound();
        }

        convite.Situacao = (evento.AcessoLivre ? EnSituacaoConvite.APROVADO : EnSituacaoConvite.AGUARDANDO);

        ConviteRepositorio.Cadastrar(convite);
        return Ok();
    }
    catch (Exception ex)
    {

        return BadRequest();
    }
}
```

INJEÇÃO DE DEPENDÊNCIA

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc().SetCompatibilityVersion(Microsoft.AspNetCore.Mvc.CompatibilityVersion.Version_2_1);

    services.AddTransient<ITipoEventoRepository, TipoEventoRepository>();

    services.AddCors(options =>
    {
        options.AddPolicy("CorsPolicy",
            builder => builder.AllowAnyOrigin()
                .AllowAnyMethod()
                .AllowAnyHeader()
                .AllowCredentials());
    });
}
```

```
private ITipoEventoRepository TipoEventoRepositorio { get; set; }
// private readonly ITipoEventoRepository TipoEventoRepositorio;

// public TiposEventosController(ITipoEventoRepository tipoEventoRepository)
public TiposEventosController()
{
    // TipoEventoRepositorio = tipoEventoRepository;
    TipoEventoRepositorio = new TipoEventoRepository();
}
```