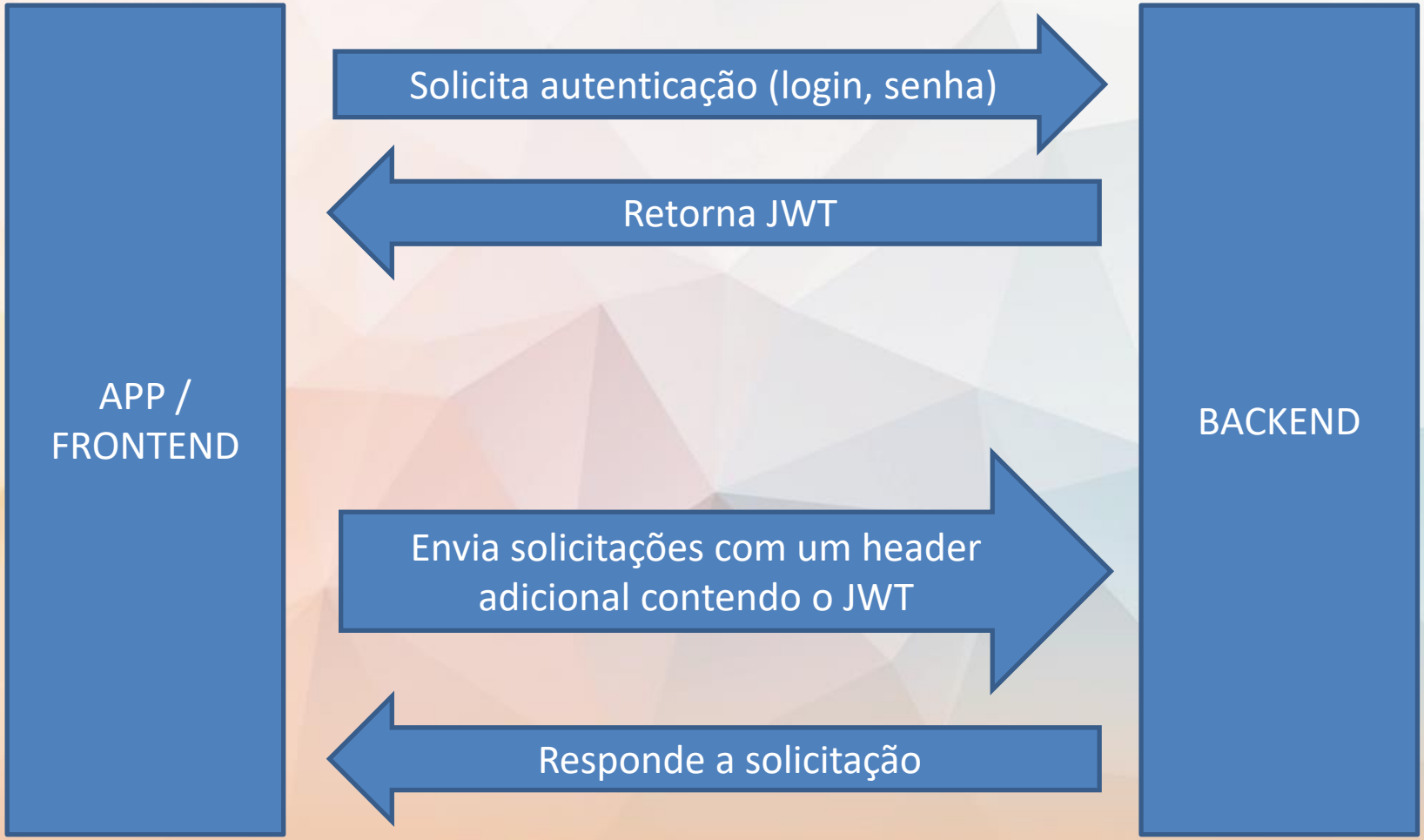
FACET
PÓS-GRADUAÇÃO

# Projeto de Banco de Dados e OO .NET

Consumindo SERVIÇOS WEB no C#
Parte 2

**Prof: Foad Mobini Kesheh**
**foadmk@gmail.com**

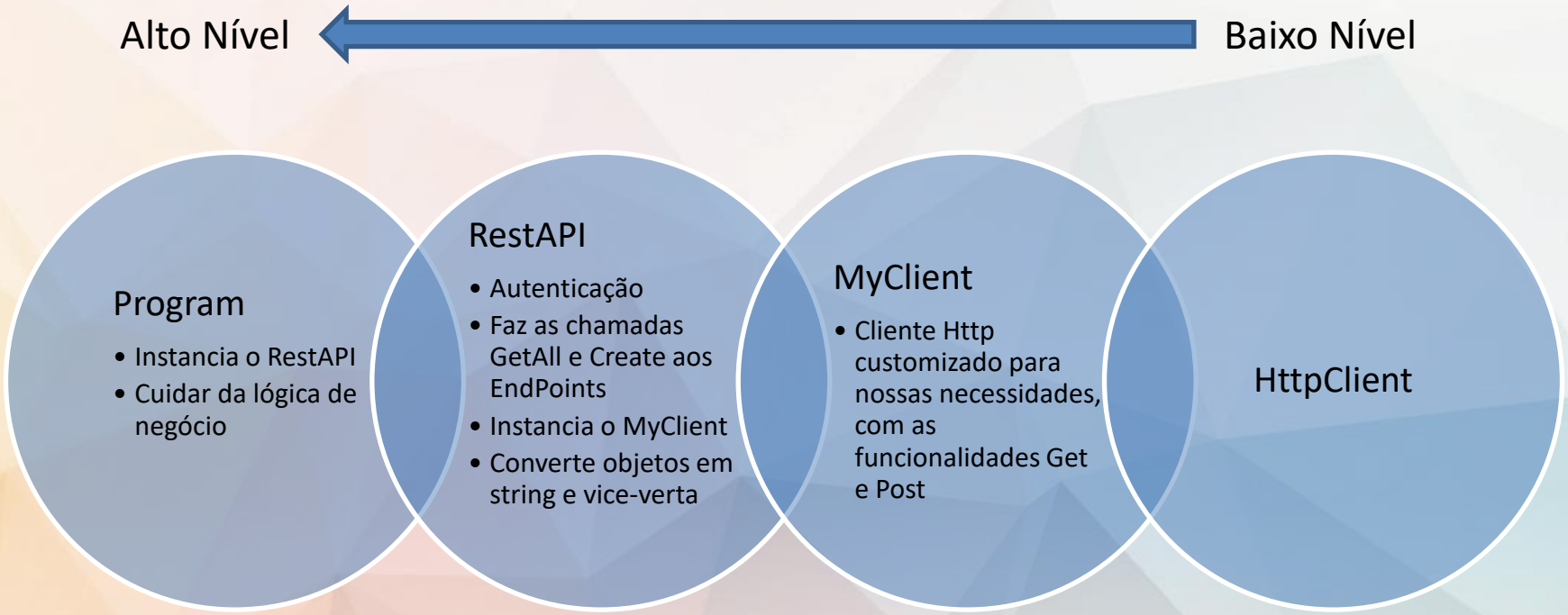
# Como funciona a autenticação em serviços WEB



# Criando um App com Login

- Solicita Login e Senha do Usuário
- Envia o Login e Senha para o Servidor e Recebe um Token JWT
- Adiciona o Token JWT ao Header das transações futuras
- Utiliza acesso com Token para adicionar e listar as Tasks do servidor backend

# Divisão de Tarefas



# Classe MyClient.cs

```
3 referências
class MyClient
{
    HttpClient client;

    1 referência
    public MyClient()
    {
        client = new HttpClient();
    }

    1 referência
    public void SetToken(string token)
    {
        client.DefaultRequestHeaders.Add("x-access-token", token);
    }
```

Instancia o HttpClient

Adiciona o Token ao Header

Note que “x-access-token” foi definido no backend em node.js. Porém, o padrão do JWT é usar a header “Authorization”

# Classe MyClient.cs

```
2 referências
Uri GetUri(string Url)
{
    return new Uri(string.Format(Url, string.Empty));
}

1 referência
ByteArrayContent ToByteArrayContent(string myContent, string headerType = "application/json")
{
    var buffer = System.Text.Encoding.UTF8.GetBytes(myContent);
    var byteContent = new ByteArrayContent(buffer);
    byteContent.Headers.ContentType = new MediaTypeHeaderValue(headerType);
    return byteContent;
}

2 referências
Exception ErrorHandler(string error)
{
    return new Exception(error);
}
```

Transforma uma Url string em um objeto Uri

Cria um http “content” a partir de uma string

Gerenciador de Erros

# Classe MyClient.cs

```
public async Task<String> Get(string restUrl)
{
    var response = await client.GetAsync(GetUri(restUrl));

    if (response.IsSuccessStatusCode)
    {
        return await response.Content.ReadAsStringAsync();
    }
    else
    {
        throw ErrorHandler($"Falha na chamada GET ao endereço: {restUrl} - StatusCode: {response.StatusCode}");
    }
}
```

Faz uma chamada GET a URL

Retorna a String de resposta

Em caso de erro, gera um Exception

# Classe MyClient.cs

Transforma o string em ByteArrayContent

```
public async Task<String> Post(string restUrl, string myContent, string headerType = "application/json")
{
    HttpResponseMessage response = await client.PostAsync(GetUri(restUrl), ToByteArrayContent(myContent, headerType));
    if (response.IsSuccessStatusCode)
    {
        return await response.Content.ReadAsStringAsync();
    }
    else
    {
        throw ErrorHandler($"Falha na chamada POST ao endereço: {restUrl} - StatusCode: {response.StatusCode}");
    }
}
```

Faz uma chamada POST a URL

Retorna a String de resposta

Em caso de erro, gera um Exception

# Classe RestAPI.cs

```
class RestAPI
{
    4 referências
    struct AuthResult
    {
        1 referência
        public string token { get; set; }
        3 referências
        public bool auth { get; set; }
    }

    2 referências
    struct AuthParams
    {
        1 referência
        public string username { get; set; }
        1 referência
        public string password { get; set; }
    }
```

Define a Struct de Resultado da API do endpoint de login

Define a Struct de parametros repassados para o endpoint de login

# Classe RestAPI.cs

```
AuthResult authResult;
MyClient client;

1 referência
public RestAPI()
{
    authResult = new AuthResult();
    authResult.auth = false;
    client = new MyClient();
}

1 referência
public bool isLogedIn()
{
    return authResult.auth;
}
```

Definir duas variáveis privadas

Construtor define os valores iniciais do authResult e instancia o MyClient

Função Auxiliar para Verificar se o RestAPI está autenticado ou não

# Classe RestAPI.cs

```
public async Task<Boolean> Auth(string RestUrl, string username, string password)
{
    AuthParams authParams = new AuthParams();
    authParams.username = username;
    authParams.password = password;

    string myContent = JsonConvert.SerializeObject(authParams);
    try
    {
        string str = await client.Post(RestUrl, myContent);
        AuthResult objs = JsonConvert.DeserializeObject<AuthResult>(str);
        client.SetToken(objs.token);
        authResult = objs;
        return objs.auth;
    }
    catch (Exception e)
    {
        throw e;
    }
}
```

Cria a string de autenticação {“username”: “foad”, password: “1234”}

Faz uma chamada POST

Decodifica a resposta

Seta o token no header do nosso MyClient

Caso o Post retornar um erro

# Classe RestAPI.cs

```
public async Task<List<T>> GetAll<T>(string RestUrl)
{
    List<T> objs;

    try
    {
        string str = await client.Get(RestUrl);
        objs = JsonConvert.DeserializeObject<List<T>>(str);
        return objs;
    }
    catch (Exception e)
    {
        throw e;
    }
}
```

Lê a string do endpoint da API

Mapeia a string numa lista de objeto genérico T

Em caso de erro, joga a exception para classe superior

# Classe RestAPI.cs

```
public async Task<Boolean> Create<T>(string restUrl, T obj)
{
    var myContent = JsonConvert.SerializeObject(obj);

    try
    {
        await client.Post(restUrl, myContent);
        return true;
    }
    catch(Exception e)
    {
        throw e;
    }
}
```

Transforma o objeto genérico T em uma string

Cria um Post com o objeto serializado

Em caso de erro, joga a exception para classe superior

# Classe Program.cs

```
class Program
{
    6 referências
    struct MyTask
    {
        0 referências
        public string _id { get; set; }
        3 referências
        public string name { get; set; }
        2 referências
        public DateTime created_date { get; set; }
        2 referências
        public string status { get; set;  }
    }
```

Primeiro declaramos o struct da nossa “Taks” para não confudir com a classe Task do C# usamos MyTaks

# Program.cs - Criar uma Task Principal

```
static RestAPI restAPI;
1 referência
static async Task<Boolean> Principal()
{
    if(!restAPI.isLogedIn()) {
        Console.Write("Login: ");
        string username = Console.ReadLine();
        Console.Write("Senha: ");
        string password = Console.ReadLine();

        await restAPI.Auth("http://localhost:3000/login", username, password);
    }

    Console.Write("Digite uma nova tarefa: ");

    MyTask t = new MyTask();
    t.name = Console.ReadLine();
    t.status = "pending";
    t.created_date = DateTime.Now;

    if (t.name.Length > 0)
    {
        await restAPI.Create<MyTask>("http://localhost:3000/tasks", t);
    }

    List<MyTask> list = await restAPI.GetAll<MyTask>("http://localhost:3000/tasks");

    Console.WriteLine("Lista de Tarefas:");
    foreach(MyTask myTask in list)
    {
        Console.WriteLine($"{myTask.name} - {myTask.status} - {myTask.created_date}");
    }
    return true;
}
```

Verificar se o RestAPI está logado

Tenta autenticar

Salva uma nova Task

Carrega uma Lista de todas Tasks

# Program.cs - Método Main

```
0 referências
static void Main()
{
    restAPI = new RestAPI();

    while (true)
    {
        Task<Boolean> x = Principal();
        while (!x.IsCompleted)
        {
            System.Threading.Thread.Sleep(50);
        }
        if (x.IsFaulted)
        {
            Console.WriteLine(x.Exception.Message);
        }
    }
}
```

Instancia RestAPI

Executa Task Principal

Aguarda a Task Completar

Em caso de falha, mostra o erro

# Exercício

- Complete a CRUD com
  - Criar usuário
  - Ver Tarefa
  - Atualizar Tarefa
  - Excluir Tarefa