

## Bootcamp IGTI

### Desafio Final

<b>Módulo 5 – Desafio Final</b>	<b>Speed Dev Wiz</b>
---------------------------------	----------------------

#### Objetivos

Exercitar os seguintes conceitos trabalhados no Curso:

- ✓ Criação de páginas utilizando HTML.
- ✓ Realizar interações com o DOM.
- ✓ Realizar requisições HTTP.
- ✓ Criação de API utilizando o ASP .NET Core.
- ✓ Realizar o Mapeamento Objeto Relacional utilizando o ORM Entity Framework Core.
- ✓ Persistência de dados utilizando o banco de dados relacional SQL Server.

#### Enunciado

Construir aplicação que simule uma biblioteca, só que essa aplicação terá apenas algumas funcionalidades de uma aplicação real, o intuito é aplicar o que foi aprendido durante todos os módulos do Bootcamp realizando a criação de um projeto de ponta a ponta. Para isso deverá ser construído duas aplicações onde a primeira será uma aplicação web usando HTML, CSS e Javascript que tenha as seguintes telas:

- Tela para realização de login na aplicação.
- Tela para realizar o cadastro/atualização de um livro. Na mesma página deverá mostrar também a listagem de todos os livros cadastrados.

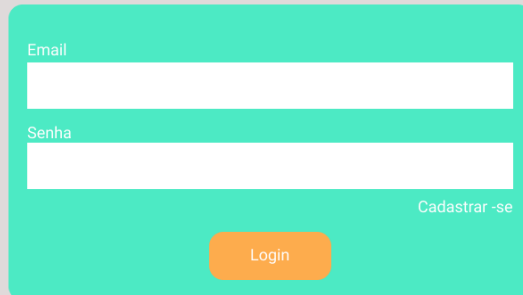
A segunda aplicação será uma Web API que irá expor os *endpoint's*. A API deverá disponibilizar as seguintes funcionalidades:

- Possibilitar realização do login.
- Possibilitar o cadastro de um novo usuário.
- Possibilitar o cadastro de um livro.
- Possibilitar a atualização dos dados de um livro existente.
- Possibilitar a exclusão de um livro.
- Possibilitar o cadastro de um autor.
- Listar os dados do livro e seu respectivo autor.

## Atividades

Os alunos deverão desempenhar as seguintes atividades:

1. Caso o aluno desejar, poderá utilizar o código fonte *base* relacionado ao projeto do **front end** que será disponibilizado na área de *Avisos do Professor* na plataforma. Esse código fonte conterá algumas funcionalidades da aplicação front end para que o aluno possa usar como base para a resolução deste desafio.
2. Criar uma página para que o usuário possa realizar o login na aplicação, essa página deve conter os campos de e-mail e senha. Para realizar o login ao clicar no botão login, o *endpoint* do tipo **POST** da controller *AuthController* deve ser chamado, utilize a função *fetch* para isso. Segue abaixo sugestão de layout.

A login form mockup with a light blue background. It features two input fields: 'Email' and 'Senha' (Password). Below the 'Senha' field is a 'Cadastrar-se' (Sign up) link. At the bottom center is an orange 'Login' button.

### Exemplo: Tela de Login.

3. Após fazer login, será apresentado ao usuário o formulário para o cadastro de um novo livro e logo abaixo do formulário de cadastro a listagem com todos os livros cadastrados no banco de dados. Para salvar um novo livro é necessário utilizar a função *fetch* e consumir o *endpoint* do tipo **POST** da controller *LivrosController*. Para buscar todos os livros cadastrados, utilizar a função *fetch* e consumir o *endpoint* do tipo **GET** da controller *LivrosController*. As informações de cada um dos livros cadastrados podem ser exibidas em um card contendo o nome do livro, o nome do autor e o ano de lançamento. Segue abaixo a sugestão do layout para a apresentação dos elementos em tela, lembrando que o layout em questão não será avaliado e sim as funcionalidades de cadastro, exclusão e listagem.

Descrição

ISBN

Autor

Selecione o Autor

▼

Ano Lançamento

Cadastrar

Excluir

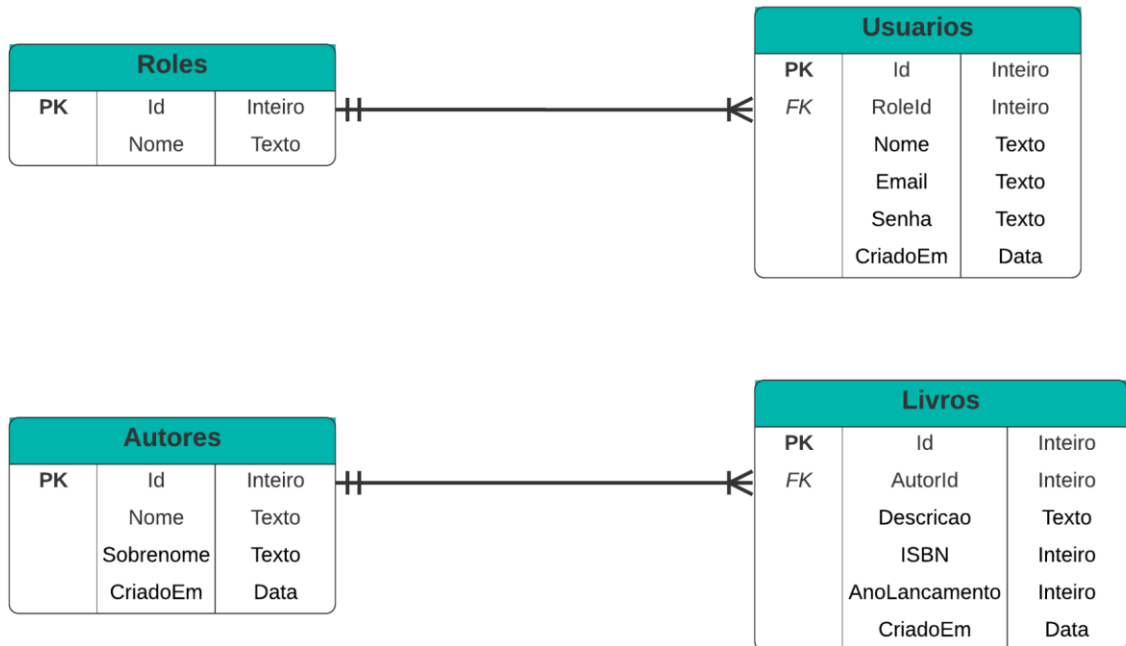
<div>DESCRICAO_LIVRO</div> <div>NOME_AUTOR</div> <div>ANO_LANCAMENTO</div>	<div>DESCRICAO_LIVRO</div> <div>NOME_AUTOR</div> <div>ANO_LANCAMENTO</div>	<div>DESCRICAO_LIVRO</div> <div>NOME_AUTOR</div> <div>ANO_LANCAMENTO</div>	<div>DESCRICAO_LIVRO</div> <div>NOME_AUTOR</div> <div>ANO_LANCAMENTO</div>	<div>DESCRICAO_LIVRO</div> <div>NOME_AUTOR</div> <div>ANO_LANCAMENTO</div>
<div>DESCRICAO_LIVRO</div> <div>NOME_AUTOR</div> <div>ANO_LANCAMENTO</div>	<div>DESCRICAO_LIVRO</div> <div>NOME_AUTOR</div> <div>ANO_LANCAMENTO</div>	<div>DESCRICAO_LIVRO</div> <div>NOME_AUTOR</div> <div>ANO_LANCAMENTO</div>	<div>DESCRICAO_LIVRO</div> <div>NOME_AUTOR</div> <div>ANO_LANCAMENTO</div>	<div>DESCRICAO_LIVRO</div> <div>NOME_AUTOR</div> <div>ANO_LANCAMENTO</div>
<div>DESCRICAO_LIVRO</div> <div>NOME_AUTOR</div> <div>ANO_LANCAMENTO</div>	<div>DESCRICAO_LIVRO</div> <div>NOME_AUTOR</div> <div>ANO_LANCAMENTO</div>	<div>DESCRICAO_LIVRO</div> <div>NOME_AUTOR</div> <div>ANO_LANCAMENTO</div>	<div>DESCRICAO_LIVRO</div> <div>NOME_AUTOR</div> <div>ANO_LANCAMENTO</div>	<div>DESCRICAO_LIVRO</div> <div>NOME_AUTOR</div> <div>ANO_LANCAMENTO</div>
<div>DESCRICAO_LIVRO</div> <div>NOME_AUTOR</div> <div>ANO_LANCAMENTO</div>	<div>DESCRICAO_LIVRO</div> <div>NOME_AUTOR</div> <div>ANO_LANCAMENTO</div>	<div>DESCRICAO_LIVRO</div> <div>NOME_AUTOR</div> <div>ANO_LANCAMENTO</div>	<div>DESCRICAO_LIVRO</div> <div>NOME_AUTOR</div> <div>ANO_LANCAMENTO</div>	<div>DESCRICAO_LIVRO</div> <div>NOME_AUTOR</div> <div>ANO_LANCAMENTO</div>

### Exemplo: Tela de Cadastro de Usuário com Listagem de Livros Cadastrados.

**Nota:** Para popular o *select* de autor deve ser utilizado a função *fetch* consumindo o *endpoint* disponibilizado: `/autores/listar` responsável por listar todos os autores cadastrados. O payload (request) para persistir, o livro no banco de dados deve conter as seguintes informações.

- Descrição
  - ISBN
  - Autor – Apenas o **value** do select selecionado. Nesse caso será o *AutorId*.
  - Ano Lançamento
4. Ao clicar em um card de um livro qualquer da listagem, as informações do livro escolhido deverão ser mostradas nos respectivos campos na parte superior da tela para que o usuário possa atualizar as informações do livro ou até mesmo excluir o livro do banco de dados.

5. Criar o banco de dados conforme a estrutura abaixo:



DER.

### Tabela Roles

- Campos:

- Id do tipo INTEGER | PK (Primary Key) | NOT NULL
- Nome do tipo VARCHAR(30) NOT NULL

### Tabela Usuarios

- Campos:

- Id do tipo INTEGER | PK (Primary Key) | NOT NULL
- RoleId do tipo INTEGER | FK (Foreign Key) | NOT NULL
- Nome do tipo VARCHAR(60) | NOT NULL
- Email do tipo VARCHAR(255) | NOT NULL

- Senha do tipo VARCHAR(60) | NOT NULL
- CriadoEm do tipo DATETIME | NOT NULL

### Tabela Livros

- **Campos:**

- **Id** do tipo INTEGER | PK (Primary Key) | NOT NULL
- *AutorId* do tipo INTEGER | FK (Foreign Key) | NOT NULL
- Descricao do tipo VARCHAR(60) | NOT NULL
- ISBN do tipo INTEGER | NOT NULL
- AnoLancamento do tipo INTEGER | NOT NULL
- CriadoEm do tipo DATETIME | NOT NULL

### Tabela Autores

- **Campos:**

- **Id** do tipo INTEGER | PK (Primary Key) | NOT NULL
- Nome do tipo VARCHAR(60) | NOT NULL
- Sobrenome do tipo VARCHAR(60) | NOT NULL
- CriadoEm do tipo DATETIME | NOT NULL

5. Criar um projeto do tipo ASP.NET Core Web API que fará todo o gerenciamento das regras de negócio da aplicação (back end). Para essa aplicação a separação em camadas não é necessário, mas seria o ideal que houvesse uma separação lógica para melhor representar a solução. A ideia principal é o gerenciamento de usuários feitos de uma forma simples, livros e autores utilizando um processo de autenticação/autorização com tokens do tipo JWT (Bearer Token); utilizaremos os verbos HTTP para representar ações CRUD e seguiremos as convenções de APIs em conjunto com os Action Results oriundos do framework. Como sugestão para as controllers temos:

## Gerenciamento de usuários:

<b>Controller</b>	UsuariosController
<b>Finalidade</b>	Cadastrar um novo usuário para que este possa acessar o sistema de acordo com as permissões definidas.
<b>Método(s)</b>	POST
<b>Endpoint Url</b>	Utilize a própria url base da controller, por exemplo: /api/v1/usuarios com o método POST associado a essa url, seguindo a convenção de APIs com essa url todos os verbos podem ser utilizados sem adicionar nenhuma palavra a rota.
<b>Ações</b>	Criar um novo usuário no banco de dados.
<b>Payload</b>	Nome ( <b>string</b> ), Email ( <b>string</b> ), Senha, ( <b>string</b> ), RoleId ( <b>Guid</b> - Papel do usuário representado na tabela de <b>Roles</b> ), CriadoEm ( <b>DateTime</b> ).
<b>Response</b>	Crie um response customizado para indicar casos de sucesso, falha ou erro. Aplique as convenções para o retorno associados ao método POST. ( <i>Verifique um possível exemplo de response customizado abaixo</i> ).

## Gerenciamento de identidade:

<b>Controller</b>	AuthController
<b>Finalidade</b>	Api responsável pela emissão de tokens de acesso, baseado nas credenciais do usuário, a geração de token insere as declarações (claims) referentes as credenciais, sendo uma delas o próprio papel do usuário. Essa informação é um importante elemento para o processo de autorização.
<b>Método(s)</b>	POST
<b>Endpoint Url</b>	Exemplo: /api/v1/auth/login
<b>Ações</b>	Gera um novo token de acesso baseado nas credenciais do usuário.

<b>Payload</b>	Email ( <b>string</b> ), Senha( <b>string</b> )
<b>Response</b>	Crie um response (objeto) com as propriedades TokenAcesso ( <b>string</b> ), Expiracao ( <b>int</b> ), TipoToken ( <b>string</b> ). Aplique as convenções para o retorno associados ao método POST.

### Gerenciamento de Livros:

<b>Controller</b>	LivrosController
<b>Finalidade</b>	Api responsável pelo gerenciamento de livros.
<b>Método(s)</b>	GET, POST, PUT, DELETE
<b>Endpoint Url</b>	Utilize a própria url base da controller, por exemplo: /api/v1/livros com os métodos supracitados associado a essa url seguindo a convenção de APIs. Todos os verbos podem ser utilizados sem adicionar nenhuma palavra a rota (HttpGet, HttpPost, HttpPut, HttpDelete)
<b>Ações</b>	Gerenciamento de livros (CRUD). O método POST será responsável por adicionar um novo livro, o PUT alterar um livro existente, DELETE representa uma operação de deleção de um livro existente e o GET vai recuperar livros baseados em critérios aos quais serão inferidos como parâmetro na action.
<b>Payload</b>	<p><b>POST:</b> Descricao (<b>string</b>), ISBN (<b>string</b>), AnoLancamento (<b>int</b>), AutorId (<b>int</b>).</p> <p><b>PUT:</b> Livroid (<b>int</b>), Descricao (<b>string</b>), AnoLancamento (<b>int</b>).</p> <p><b>DELETE:</b> Livroid (<b>int</b>).</p> <p><b>GET:</b> NomeLivro (<b>string</b>), ISBN (<b>string</b>), AnoLancamento (<b>int</b>).</p>
<b>Response</b>	Crie um response customizado para indicar casos de sucesso, falha ou erro. Aplique as convenções para o retorno associados aos métodos. Para o método <b>GET</b> retorne as



	propriedades DescricaoLivro ( <b>string</b> ), ISBN ( <b>string</b> ), AnoLancamento ( <b>int</b> ), NomeAutor ( <b>string</b> ). (Verifique um possível exemplo de response customizado abaixo).
--	---

### Gerenciamento de Autores:

<b>Controller</b>	AutoresController
<b>Finalidade</b>	O método <b>POST</b> será responsável pelo cadastro de Autores, o <b>GET</b> vai recuperar todos os registros de autores cadastrados no banco de dados.
<b>Método(s)</b>	GET, POST
<b>Endpoint Url</b>	Utilize a própria url base da controller, por exemplo: /api/v1/autores com o método POST associado a essa url, seguindo a convenção de APIs com essa url todos os verbos podem ser utilizados sem adicionar nenhuma palavra a rota.
<b>Ações</b>	Crie um novo autor no banco de dados.
<b>Payload</b>	<b>POST:</b> Nome ( <b>string</b> ), Sobrenome( <b>string</b> ).
<b>Response</b>	Crie um response customizado para indicar casos de sucesso, falha ou erro. Aplique as convenções para o retorno associados ao método <b>POST</b> . Para o método <b>GET</b> , retorne as propriedades AutorId ( <b>int</b> ), Nome ( <b>string</b> ), Sobrenome ( <b>int</b> ). (Verifique um possível exemplo de response customizado abaixo).

Para responses podemos customizá-los de acordo com as nossas preferências. Alguns exemplos são representados abaixo:

**Sucesso** (200, 201, 204)

```
{
  "Status": "Sucesso",
  "Code": "201"
}
```

```
"Data" : true
}
```

**Sucesso (200, 201, 204)**

```
{
  "Status" : "Sucesso",
  "Code" : "201"
  "Data" : [{
    Livro1,
    Livro2
  }]
}
```

**Falha (400, 404)**

```
{
  "Status" : "Falha",
  "Code" : "400"
  "Data" : "Não encontrado"
}
```

**Falha (400, 404)**

```
{
  "Status" : "Falha",
  "Code" : "404"
  "Data" : {
    "Propriedade" : "Nome",
    "Mensagem" : "Nome deve ser informado."
  }
}
```

**Erro (500)**

```
{
  "Status" : "Erro",
  "Code" : "500"
  "Mensagem" : "Falha ao recuperar dados."
}
```

- Para o processo de autenticação e autorização, ative os middlewares responsáveis.

- Implemente as soluções utilizando o JWT do pacote ***Microsoft.AspNetCore.Authentication.JwtBearer*** e configure as dependências através no container.
- Defina as configurações para a geração do token através do arquivo de configuração ***appSettings.json***.
- Para as roles do usuário, não utilizaremos controllers, estas informações serão criadas como seed do banco de dados. Como sugestão podemos utilizar 2 roles, sendo uma representar os papéis de administrador, com o nome “admin” e para usuários comuns podemos utilizar “comum”.
- Decore as controllers/actions com os atributos de autorização.
- Compare as roles aplicadas na controller com as roles que serão parte do token de acesso.
- As operações de cadastro de usuário e autores só podem ser acessadas pela role “admin”, as operações de autenticação através do endpoint de login serão de acesso público “AllowAnonymous” e as demais por todos as roles “admin” e “comum”.

