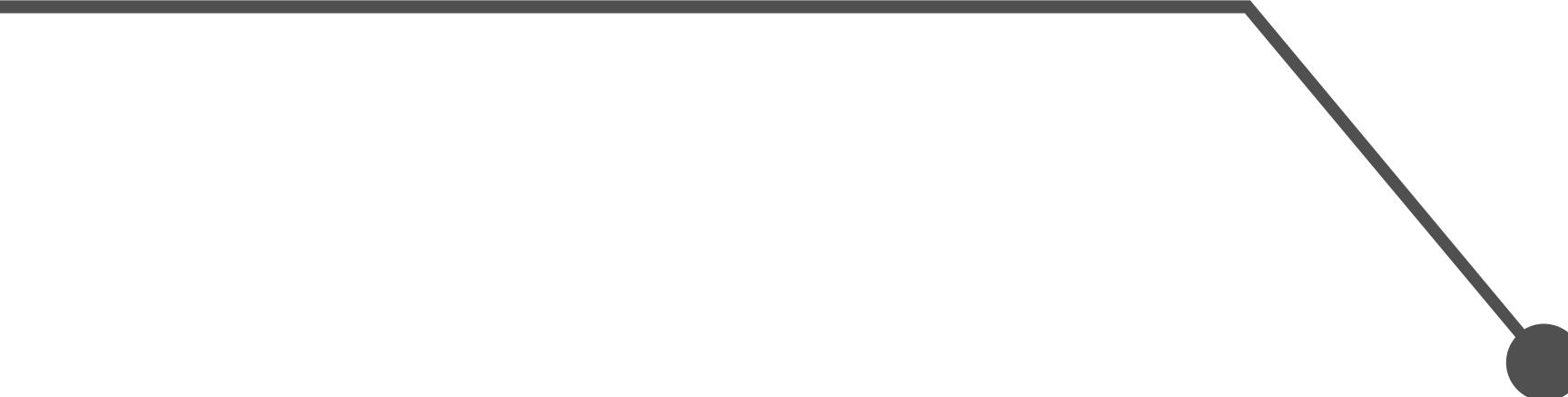


---

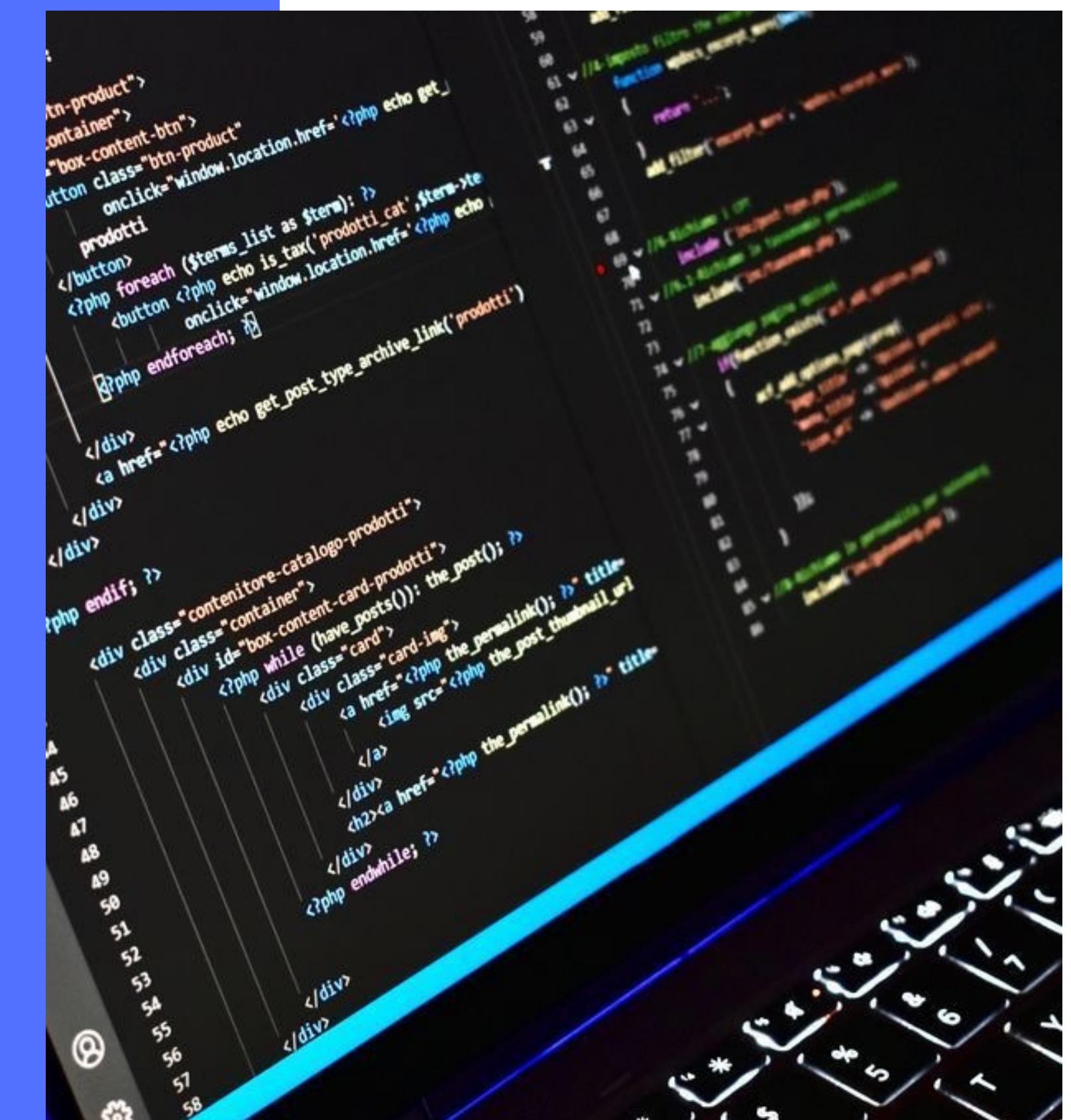


# **PROJETO**

# **3**



# TECNOLOGIAS



A screenshot of a code editor showing a file with PHP code. The code includes HTML templates and database queries. A blue cursor is visible in the editor. The bottom right corner shows a portion of a black computer keyboard.

```
<tn-product>
  <container>
    <box-content-bth>
      <button class="btn-product" onclick="window.location.href='?php echo get_permalink($post->post_name)'>Prodotti</button>
      <php foreach ($terms_list as $term): ?>
        <button <?php echo is_tax('prodotti_cat', $term->term_id) ?> onclick="window.location.href='?php echo get_permalink($term->term_id)'> <?php endforeach; ?>
      </div>
      <a href="?php echo get_post_type_archive_link('prodotti')">
        <div>
          <div class="contenitore-catalogo-prodotti">
            <div class="container">
              <div id="box-content-card-prodotti">
                <php while (have_posts()): the_post(); ?>
                  <div class="card">
                    <div class="card-img">
                      <a href="?php the_permalink(); ?> 
                    </div>
                    <h2><php the_title(); ?></h2>
                  </div>
                <php endwhile; ?>
              </div>
            </div>
          </div>
        </div>
      </a>
    </div>
  </box-content-bth>
</container>
<tn-product>
```

# CLEAN ARCHITECTURE

O código é organizado em camadas com responsabilidades específicas, facilitando a manutenção, teste e escalabilidade do sistema.

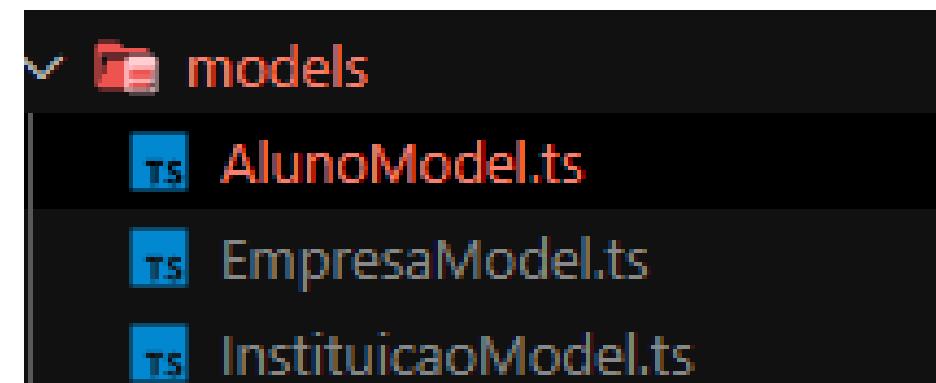
- Entities: Contém as entidades de negócios da aplicação
- Models: Contém os modelos de dados
- UseCase: Esta camada é onde a lógica de negócios da aplicação reside.
- Routes: Contém as definições das rotas da aplicação
- Repository: Responsável pela interação com o banco de dados ou qualquer outro mecanismo de armazenamento de dados.

# ARQUITETURA MVC



- SEPARAR A LÓGICA DE NEGÓCIOS DA APRESENTAÇÃO
- MODEL
- VIEW
- CONTROLLER
- MVC FACILITA A MANUTENÇÃO E ESCALABILIDADE DO CÓDIGO

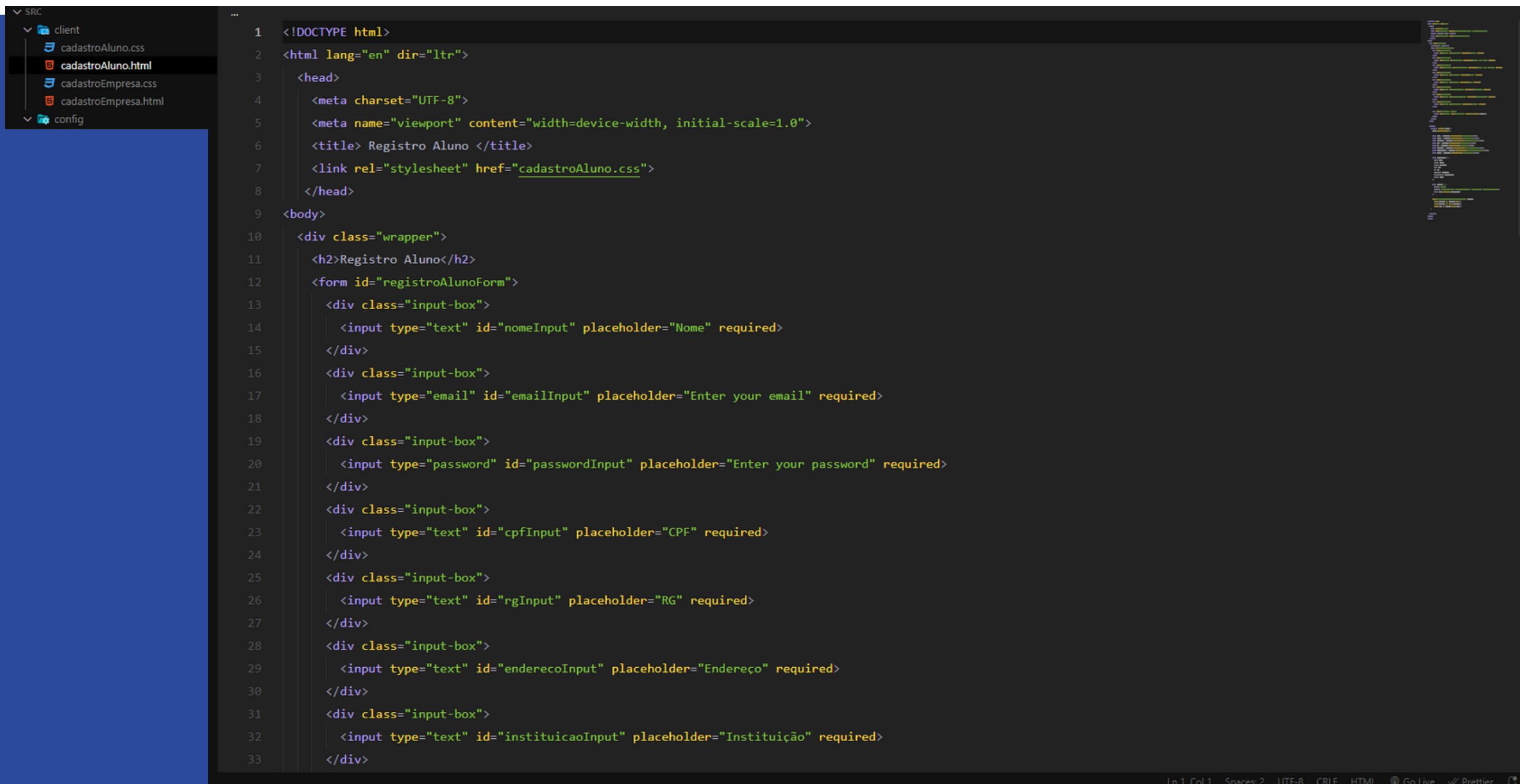
# MODELS



```
 3 import { InstituicaoModel } from './instituicaoModel';
 4
 5 export class AlunoModel extends Model {
 6   declare nome: string;
 7   declare rg: string;
 8   declare cpf: string;
 9   declare email: string;
10   declare senha: string;
11   declare endereco: string;
12   declare instituicao: string;
13   declare saldoDeMoedas: number;
14   declare curso: string;
15 }
16
17 AlunoModel.init(
18   {
19     nome: {
20       type: new DataTypes.STRING(128),
21       allowNull: false,
22     },
23     rg: {
24       type: new DataTypes.STRING(20),
25       allowNull: false,
26     },
27     cpf: {
28       type: new DataTypes.STRING(14),
29       allowNull: false,
30       unique: true
31     }
32   }
33 )
```



# VIEW



The screenshot shows a code editor with a dark theme. On the left, there is a file tree with the following structure:

- SRC
  - client
    - cadastroAluno.css
    - cadastroAluno.html**
    - cadastroEmpresa.css
    - cadastroEmpresa.html
  - config

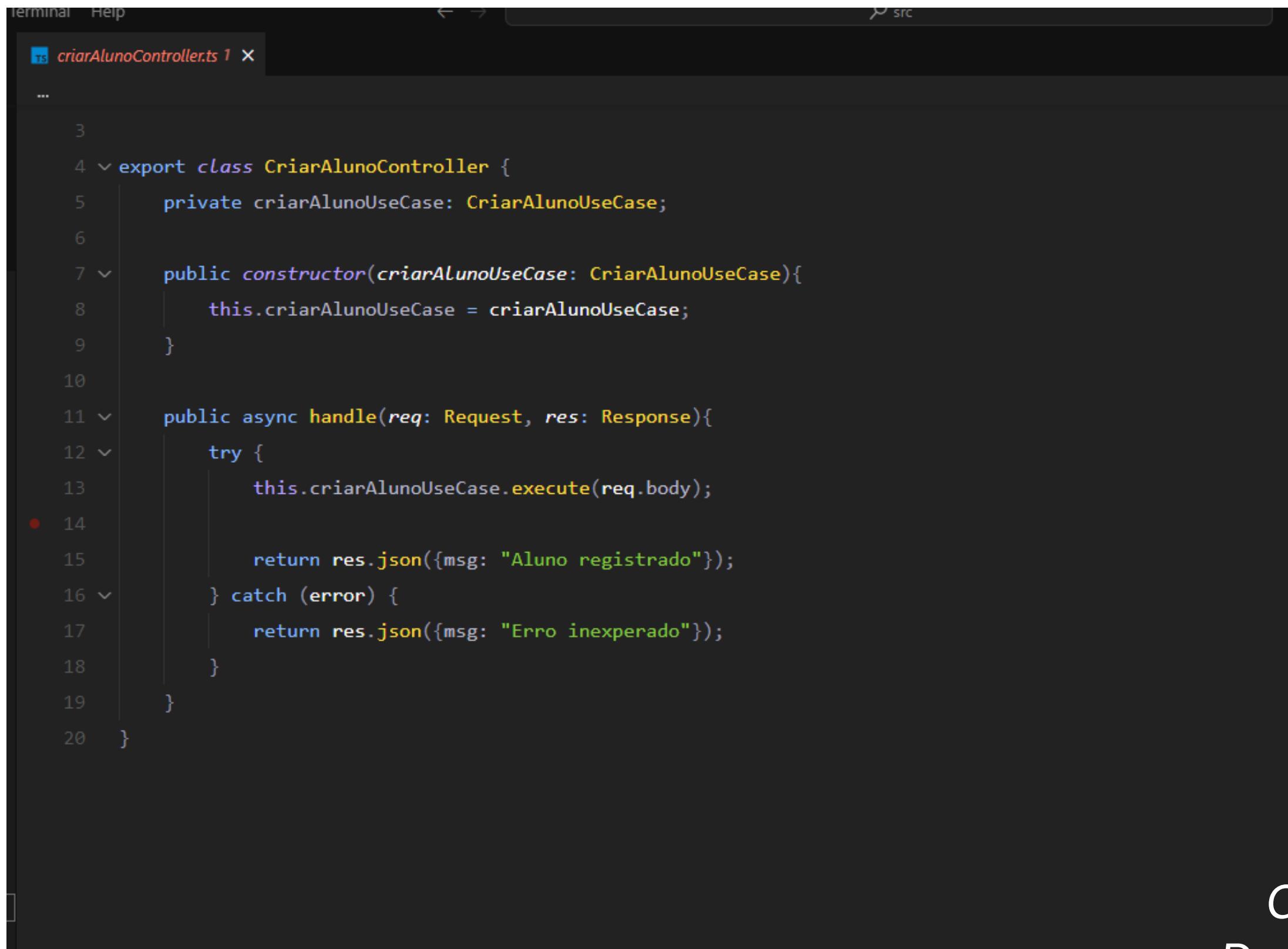
The main pane displays the content of the `cadastroAluno.html` file:

```
1 <!DOCTYPE html>
2 <html lang="en" dir="ltr">
3   <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <title> Registro Aluno </title>
7     <link rel="stylesheet" href="cadastroAluno.css">
8   </head>
9   <body>
10    <div class="wrapper">
11      <h2>Registro Aluno</h2>
12      <form id="registroAlunoForm">
13        <div class="input-box">
14          <input type="text" id="nomeInput" placeholder="Nome" required>
15        </div>
16        <div class="input-box">
17          <input type="email" id="emailInput" placeholder="Enter your email" required>
18        </div>
19        <div class="input-box">
20          <input type="password" id="passwordInput" placeholder="Enter your password" required>
21        </div>
22        <div class="input-box">
23          <input type="text" id="cpfInput" placeholder="CPF" required>
24        </div>
25        <div class="input-box">
26          <input type="text" id="rgInput" placeholder="RG" required>
27        </div>
28        <div class="input-box">
29          <input type="text" id="enderecoInput" placeholder="Endereço" required>
30        </div>
31        <div class="input-box">
32          <input type="text" id="instituicaoInput" placeholder="Instituição" required>
33        </div>
```

At the bottom of the editor, there is a status bar with the following information:

Ln 1, Col 1 Spaces: 2 UTF-8 CRLF HTML ⚡ Go Live ⚡ Prettier

# CONTROLLER



The screenshot shows a code editor window with a dark theme. At the top, there's a navigation bar with tabs for 'terminal' and 'Help'. Below the navigation bar, the title bar displays the file name 'criarAlunoController.ts' with a count of '1 X'. The main area of the editor contains the following TypeScript code:

```
terminal  Help
          ↶ ↷ ⌂ src
TS criarAlunoController.ts 1 X

...
3
4 export class CriarAlunoController {
5     private criarAlunoUseCase: CriarAlunoUseCase;
6
7     public constructor(criarAlunoUseCase: CriarAlunoUseCase){
8         this.criarAlunoUseCase = criarAlunoUseCase;
9     }
10
11    public async handle(req: Request, res: Response){
12        try {
13            this.criarAlunoUseCase.execute(req.body);
14
15            return res.json({msg: "Aluno registrado"});
16        } catch (error) {
17            return res.json({msg: "Erro inesperado"});
18        }
19    }
20}
```

The code defines a class 'CriarAlunoController' that injects a 'CriarAlunoUseCase' dependency. It has a constructor to set this dependency and a method 'handle' that takes a request and response, executes the use case with the request body, and returns a JSON response indicating success or an unexpected error.

C  
D

# CLEAN ARQUITETURE

- **Ênfase na Separação de Preocupações:** A Clean Architecture enfatiza fortemente a separação de preocupações, mantendo as regras de negócios independentes de detalhes de infraestrutura.
- **Flexibilidade e Independência:** É altamente flexível e independente de frameworks específicos, sendo adequada para diversos contextos de aplicação.
- **Camadas Claras:** Organiza o código em camadas bem definidas, como entidades, modelos, repositórios, rotas e casos de uso.
- **Foco nas Regras de Negócios:** A lógica de negócios é centralizada na camada de casos de uso, com controladores específicos para cada caso

# MVC

- **Separação de Responsabilidades:** Divide o aplicativo em três componentes principais - Modelo (Model), Visão (View) e Controlador (Controller) - para separar responsabilidades.
- **Ênfase em Interfaces Gráficas do Usuário:** É frequentemente usado em aplicativos com interfaces gráficas do usuário, onde a clareza na separação entre Modelo, Visão e Controlador é fundamental.
- **Interativo e Orientado a Eventos:** Projetado para lidar com aplicativos interativos que respondem a eventos do usuário.
- **Comum em Aplicações Web:** É amplamente utilizado em aplicativos da web, onde a visão (View) é responsável por apresentar a interface do usuário.

# MODELO RELACIONAL

- 1 A camada de persistência é responsável pelo armazenamento e recuperação de dados em um banco de dados.  
  
Para manter um código organizado, é comum criar pastas separadas para a implementação da camada de persistência.  
Cada entidade do sistema é representada por uma pasta correspondente.  
  
As interfaces definem contratos que as classes de implementação devem seguir.
- 2
- 3 Para cada entidade, temos uma classe de implementação chamada "NomeDaEntidadeRepository" que implementa a interface correspondente  
Usamos a classe "MySQLRepository" para estabelecer a conexão e realizar operações no banco de dados MySQL.

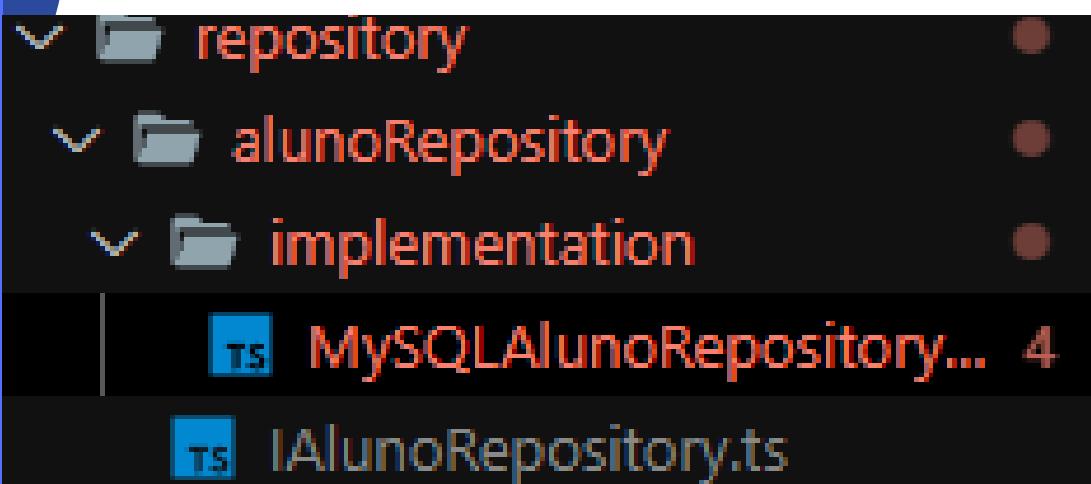




# EXEMPLO

ENTIDADE: USUÁRIO  
INTERFACE: IUSUARIOREPOSITORY  
IMPLEMENTAÇÃO: USUARIOREPOSITORY  
CLASSE DE ACESSO A BANCO DE DADOS:  
MYSQLREPOSITORY  
ESSA ESTRUTURA PROPORCIONA UMA  
SEPARAÇÃO CLARA ENTRE A LÓGICA DE  
NEGÓCIOS E A CAMADA DE  
PERSISTÊNCIA, TORNANDO O SISTEMA  
MAIS ORGANIZADO E FACILITANDO A  
MANUTENÇÃO E ESCALABILIDADE.

# EXEMPLO



```
1 import { Aluno } from "../../entities/Aluno";
2
3 export interface IAlunoRepository {
4     encontrarPeloCpf(cpf: string): Promise<Aluno | null>;
5
6     criarNovo(aluno: Aluno): Promise<void>;
7
8     remover(aluno: Aluno): Promise<void>;
9
10    atualizar(cpf: Aluno, novoAluno: Aluno): Promise<void>;
11
12    encontrarTodos(): Promise<Aluno[]>;
13}
```

```
async criarNovo(aluno: Aluno): Promise<void> {
    // throw new Error("Method not implemented.");
}

const instituicao = aluno.getInstituicao();

await AlunoModel.create({
    nome: aluno.getNome(),
    rg: aluno.getRg(),
    cpf: aluno.getCpf(),
    email: aluno.getEmail(),
    senha: aluno.getSenha(),
    endereco: aluno.getEndereco(),
    instituicao: instituicao.getNome(),
    saldoDeMoedas: aluno.getSaldoDeMoedas(),
    curso: aluno.getCurso(),
});

}

async remover(aluno: Aluno): Promise<void> {
    // throw new Error("Method not implemented.");
}

const verificarAluno = await AlunoModel.findOne({ where: { cpf: aluno.getCpf() } });

if(verificarAluno){
    await verificarAluno.destroy();
}
```