**FUNDAÇÃO OSWALDO ARANHA**

**CENTRO UNIVERSITÁRIO DE VOLTA REDONDA**

**CURSO DE SISTEMAS DE INFORMAÇÃO**

**RELATÓRIO TÉCNICO-CIENTÍFICO**

**SISTEMA DE CONTROLE PARA AVALIAÇÕES DE PROVAS E GERAÇÃO DE RELATÓRIOS DO NADD**

Autor(es)

**CAIO DOMINGUES DA SILVA SANTOS**

**HIGOR AUGUSTO DA SILVA SANTOS**

**JULIANO FERREIRA RODRIGUES**

**THOBIAS VICENTE DE SOUZA**

Orientador(es)

**ADILSON GUSTAVO DO ESPÍRITO SANTO**

**BRUNA CASIRAGHI**

**CARLOS EDUARDO COSTA VIEIRA**

**ROSENCLEVER LOPES GAZONI**

**VENÍCIO SIQUEIRA FILHO**

**Volta Redonda**

**Setembro/2019**

**SUMÁRIO**

[1. Banco de Dados 6](#_Toc18936742)

[1.1. Programação Orientada a Objetos 6](#_Toc18936743)

[1.1.1. Abstract Factory 8](#_Toc18936744)

[1.1.2. ASP.NET 10](#_Toc18936745)

[1.1.2.1. Arquitetura MVC 11](#_Toc18936746)

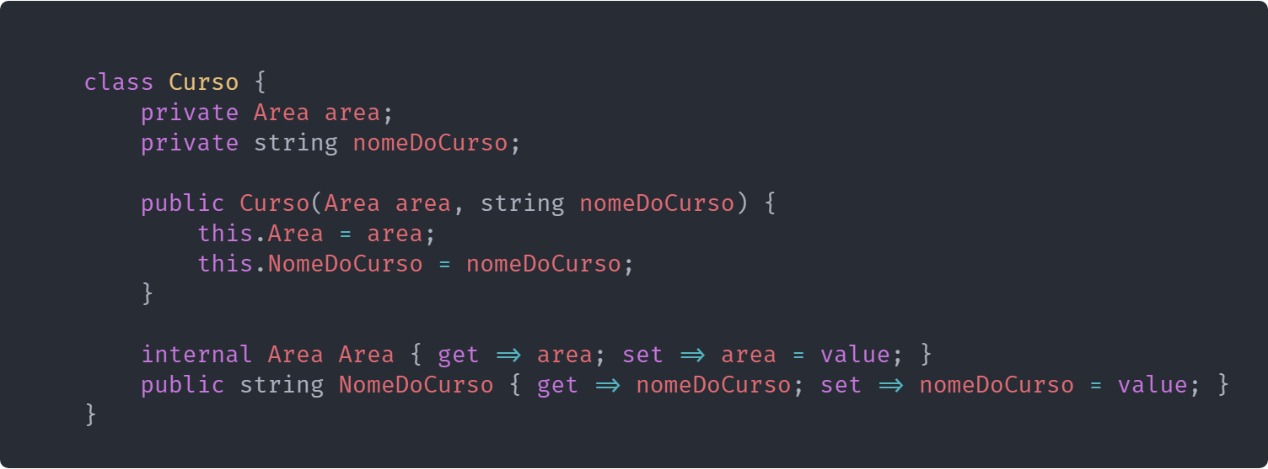
[1.1.2.2. Razor Web-Engine 11](#_Toc18936747)

[1.2. Banco de Dados 11](#_Toc18936748)

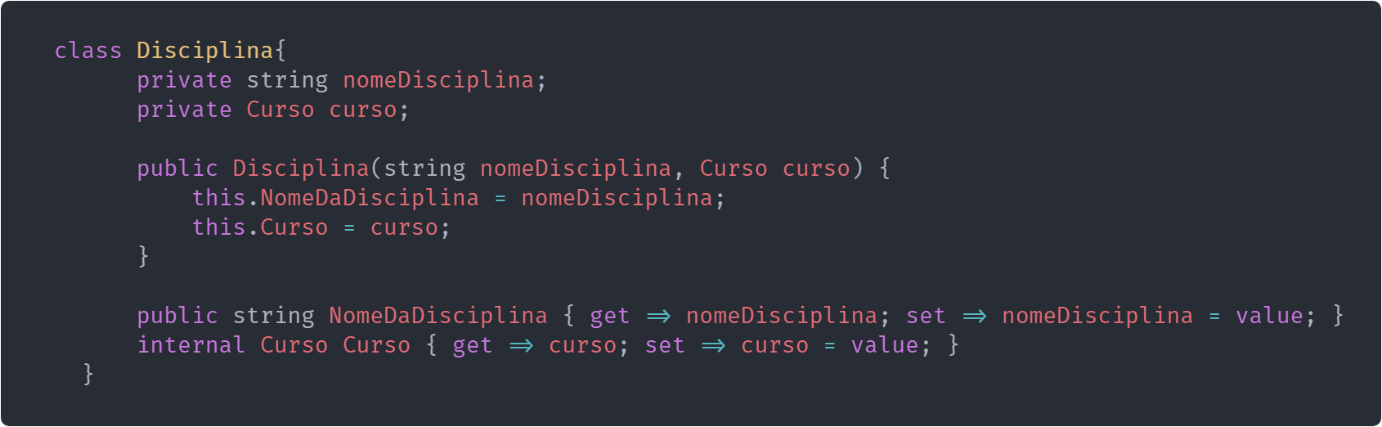
1. Banco de Dados

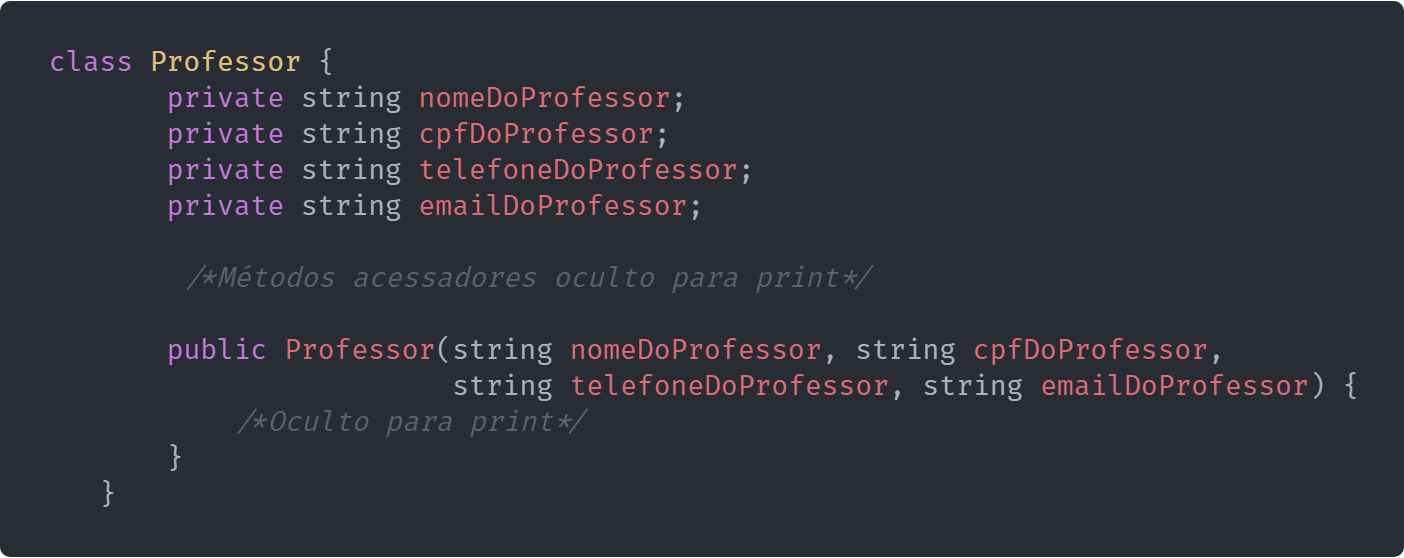
# Programação Orientada a Objetos

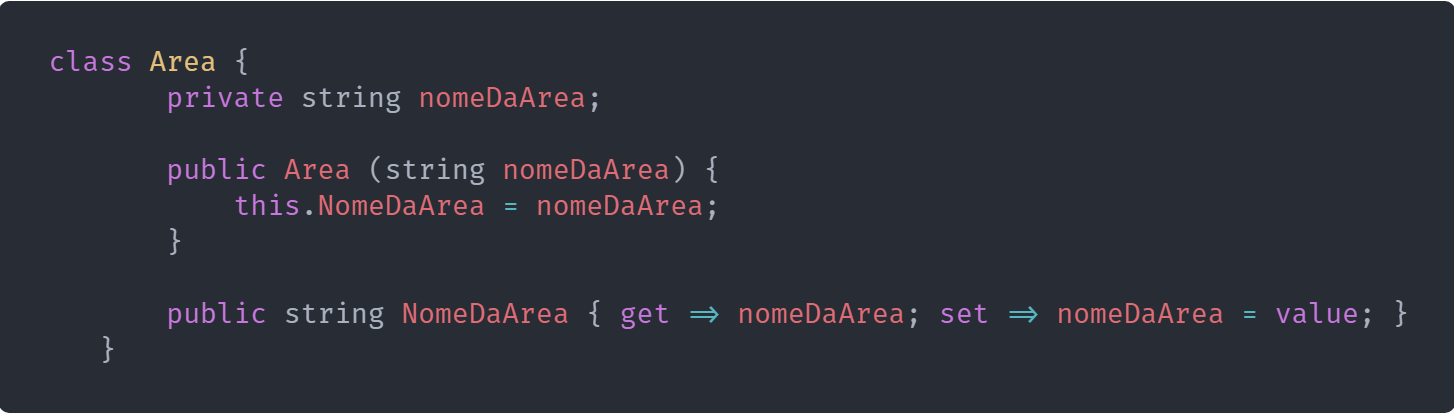
Desenvolvimento da aplicação em C#. Este é o protótipo elaborado das classes da aplicação, bem como comunicação e interconexão entre as mesmas, note que estes foram utilizados como base para a elaboração do sistema em ASP.NET, o qual resultará no objetivo final proposto ao cliente, sendo assim, a base estrutural das classes, que posteriormente serão tratadas como “Modelos” ou “Entidades” do sistema, são:







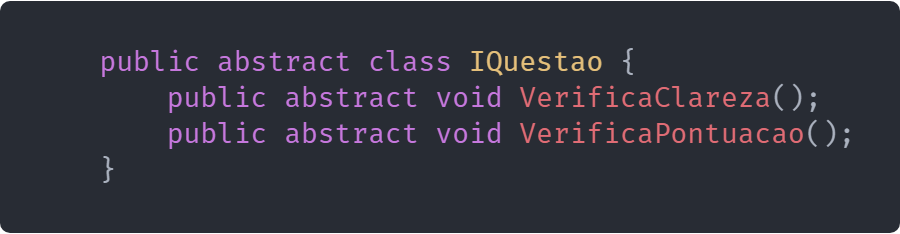


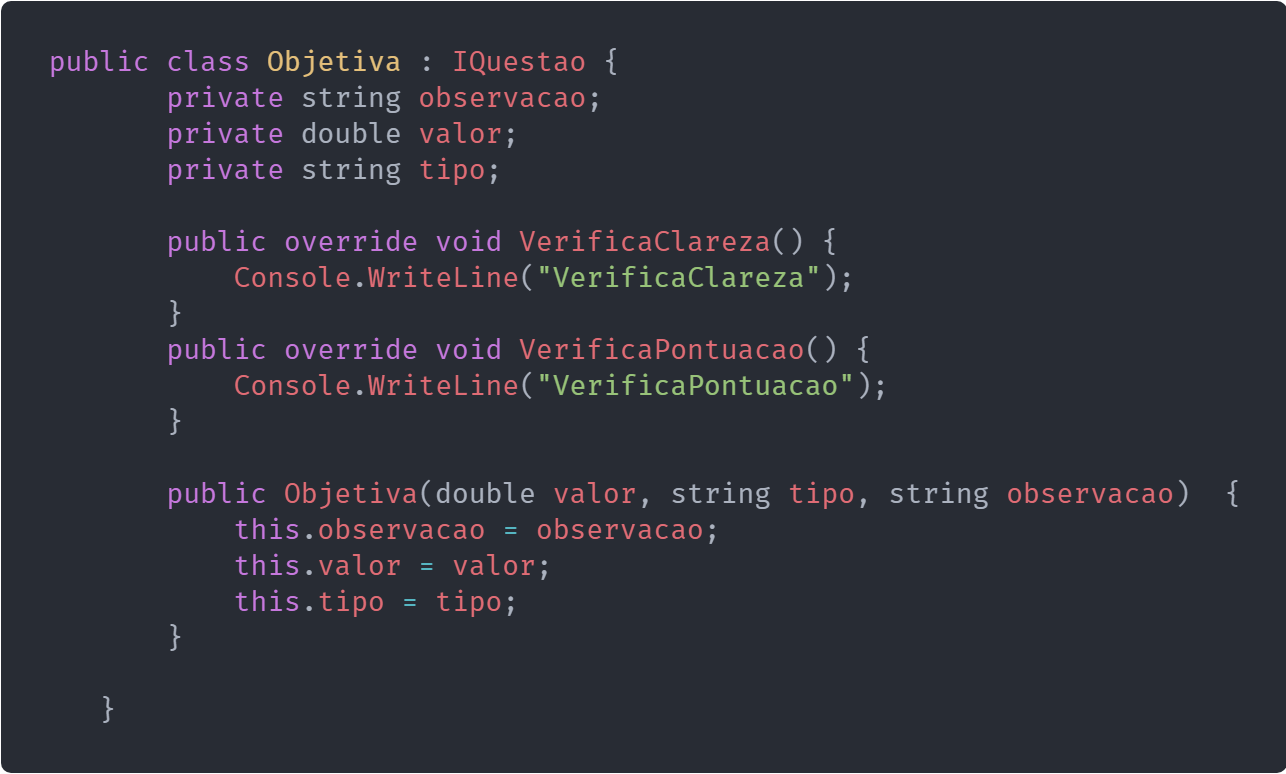
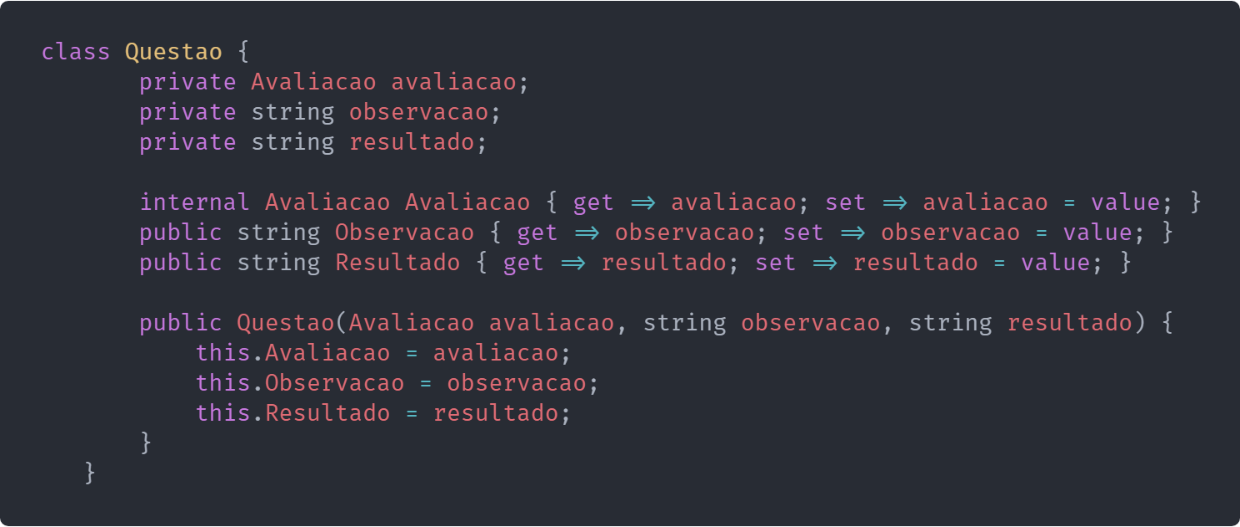
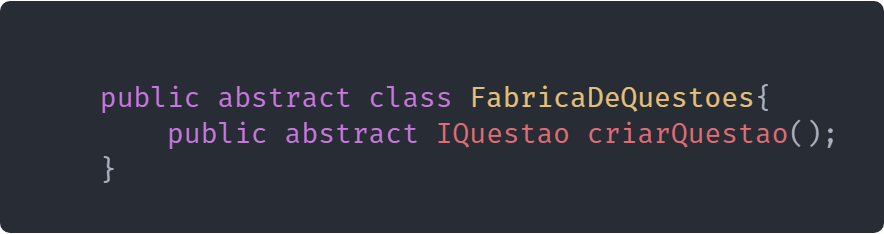
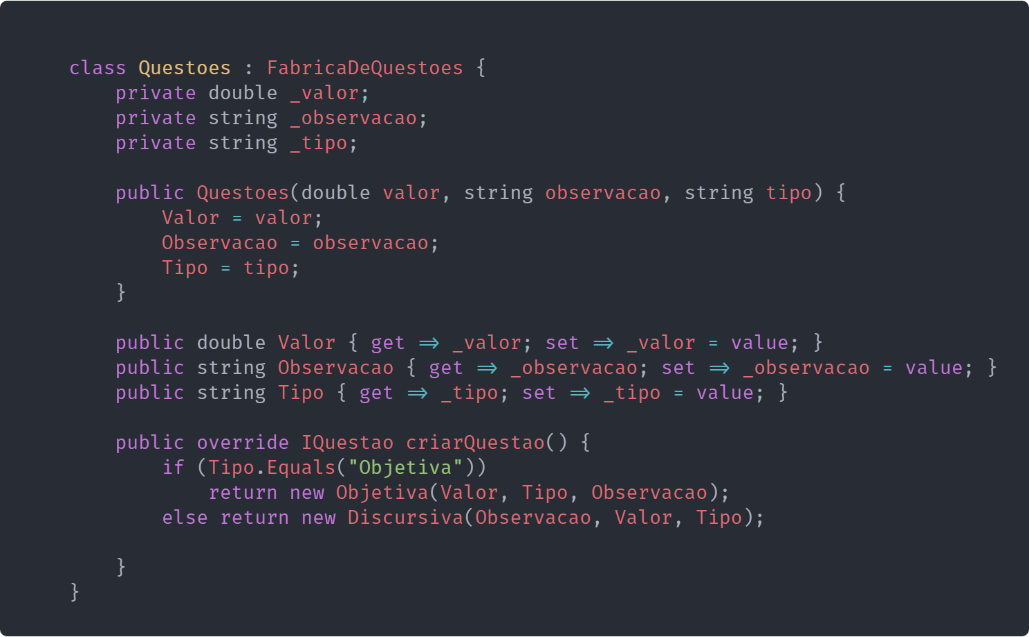
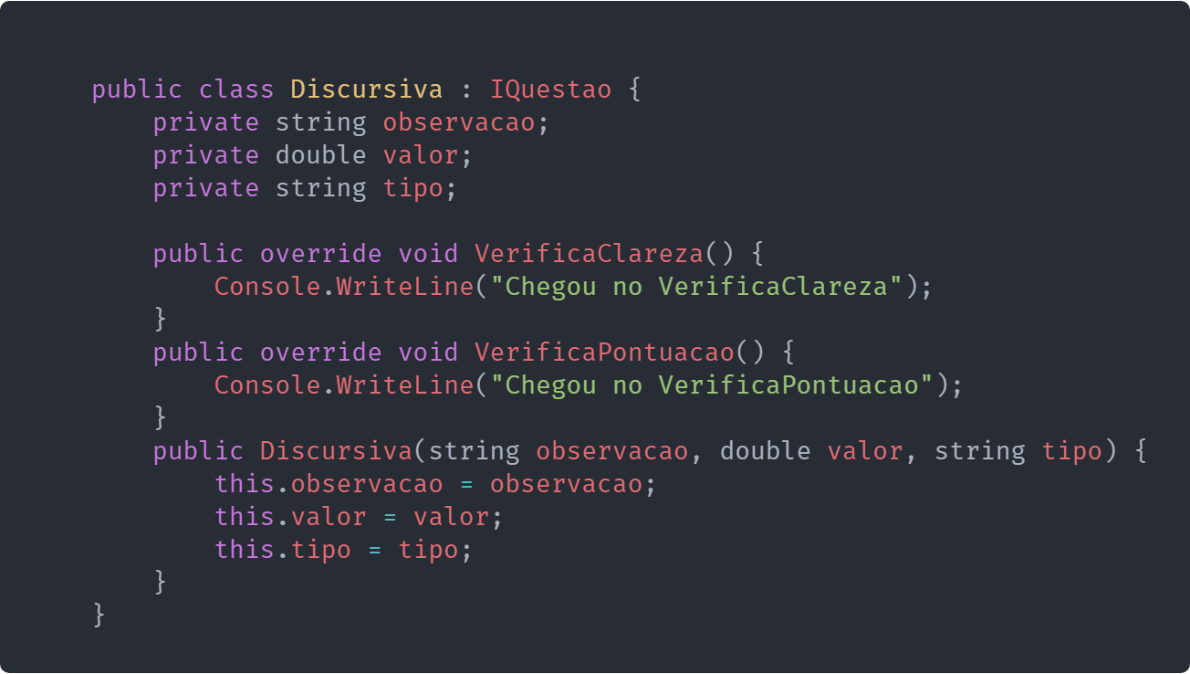


## Abstract Factory

Para a conceituação da utilização de Abstract Factory deste projeto, utilizaremos como base o objeto Questão. Uma avaliação possui **N** questões, sendo que estas podem várias dentre seu tipo: discursiva, objetiva, relacional etc.

A aplicação do conceito de Abstract Factory encaixa-se visto que podemos criar uma interface ou classe abstrata de Questão e uma Fábrica Abstrata dela, derivando destes suas respectivas Fábricas e Questões por tipo.



## ASP.NET

Desenvolvimento da aplicação, utilizando o Framework ASP.NET com as técnicas de *Scaffolding,* seguindo os padrões MVC (Movel -View – Controller) amplamente utilizados atualmente pelas aplicações web para o desenvolvimento de ferramentas, *web apps*, *progressive web applications* e *single page applications*, de forma a garantir um fluxo de informações seguro, ágil e estruturado. Buscando atender as necessidades e requisitos previamente levantados em conjunto com o cliente.

As classes utilizadas no protótipo base, foram reutilizadas como Modelo/Entidade da aplicação *web*, para minimizar a repetição de conteúdo, o código está acessível de maneira aberta, tanto o protótipo quanto a aplicação final – ainda em desenvolvimento – no GitHub: <https://github.com/caiodomingues/Projeto-NADD>.

## Arquitetura MVC

A arquitetura *Model View Controller* é uma das mais famosas para a construção de aplicações web, principalmente por apresentar uma camada de segurança na manipulação de dados quanto por ser ágil, já que o trabalho pesado fica ao lado do servidor.

Desta forma, separadamente:

* ***Model*:** parte responsável pela estrutura de dados, é a classe que dispõe a informação acerca dos dados que serão tratados pela aplicação, bem como faz as conexões com outras classes (relacionamentos). Fazer as *queries* do banco de dados dentro dos modelos é uma boa prática, não só de segurança, mas também de performance e tipagem, pois assim, ao referenciar esta classe, obtêm-se acesso aos métodos inerentes da mesma, podendo retornar os resultados da consulta sem ter de declarar várias vezes o mesmo código, aplicando então o conceito DRY.
* ***View:*** parte de retorno ao usuário, a parte visual é tratada pela Razor Web-Engine, desta forma, o ASP.NET fornece *Server Side Render*, fazendo com que as *views* tornem-se dinâmica, por exemplo:
  + O usuário **A** acessa a página inicial, que mostra seu nome;
  + O usuário **B** acessa a **mesma** página, que mostra B ao invés de A;
* ***Controller***: a principal peça de controle do sistema, quase todas as requisições vão para o controlador, por meio do *Router*, dentro deste, alguma ação é tomada, um valor é gerado ou um resultado é retornado à *View*, um fluxo completo, por exemplo, é:
  + O usuário A acessa a rota de “alterar perfil” por meio de uma requisição do tipo **GET**, o *Router* leva ao *controller*, que retorna a página para tal rota.
  + O usuário A altera os dados e salva as informações, por meio de uma requisição do tipo **POST**, agora, o controlador pega as informações inseridas pelo usuário, valida-as de acordo com o modelo, e caso estas estejam devidamente corretas, altera os dados desejados.
  + O usuário A é redirecionado pelo controlador, como retorno da ação, para a página de “perfil”, pode-se ver que suas alterações foram realizadas com sucesso.

## Razor Web-Engine

É um motor de renderização e criação de páginas web que “transcompila” informações de C# e/ou VB.NET para HTML. Foi criado especialmente para o modelo MVC 3 e o kit de ferramentas do WebMatrix, sendo posteriormente aplicado como um componente do AspNetWebStack (cadeia de ferramentas para desenvolvimento em ASP.NET), tornando-se parte do ASP.NET Core.

Razor aplica os mesmos ideais que outros *renders* ou compiladores de outras linguagens, como Blade, Twig, JSX, EJS etc: reduzir a quantidade de código HTML, aproximar o código HTML do C#, facilitar a dinamicidade das *Views* e otimizar o código.

# Banco de Dados

|  |
| --- |
| 1) Subconsulta simples (com apenas uma tabela) |
| 2) Subconsulta complexa (completando duas tabelas) |
| 3) Subconsulta utilizando a cláusula ALL |
| 4) Subconsulta utilizando a cláusula ANY |
| 5) Subconsulta utilizando a cláusula IN |
| 6) Consulta utilizando Right Join |
| 7) Consulta utilizando Full Outer Join |
| 8) Dois blocos PL/SQL que façam uso de variáveis, da cláusula %TYPE e do bloco Exception |



|  |
| --- |
| 1) Alterações e PL/SQL |
| 2) Function e Bloco Anônim |

