



Avernus Games - WebApi

Contribuições do Chat-GPT

[Avernus Games - ClassDiagram.pdf](#)

Modelagem

Após a escrita de todas as Models e suas relações, o GPT fez alguns construtores para elas.

[Chat completo - Modelagem](#)

Banco de Dados e Mapeamento

Após a aula sobre a conexão com o banco de dados, surgiram algumas dúvidas relacionadas a detalhes específicos do banco que escolhemos usar (MySQL).

Dúvidas como qual package do Entity Framework instalar e qual connection string era necessária. (Foi necessária a ajuda do professor pra identificar um erro de versão na connection string).

Durante o mapeamento das Models pro EF, inúmeras dúvidas surgiram pois nós estávamos tentando usar o mesmo padrão do Hibernate (Java), no qual é necessário explicitar todas as relações. Fora dúvidas relacionadas as entidades com herança.

Por fim, o GPT nos ajudou a chegar à um resultado satisfatório me mostrando um método do EF chamado OnModelCreating

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    base.OnModelCreating(modelBuilder);

    modelBuilder.Entity<Game>().ToTable("Game");
}
```

```
modelBuilder.Entity<RPGGame>().ToTable("RPGGame");  
modelBuilder.Entity<Vestimenta>().ToTable("Vestimenta");  
}
```

Apenas com este método, já foi possível deixar todas as tabelas e relações do banco da forma que foi idealizada.

Logo após a concepção do escopo, o GPT auxiliou com algumas dúvidas relacionadas às boas práticas da linguagem; coisas como utilizar atributos privados ou propriedades públicas. A IA explicou que quando se usa propriedades públicas em C#, a linguagem automaticamente cria atributos negativos para armazenar os dados destas propriedades e garantir o encapsulamento.

Resumo das dúvidas sanadas pela IA:

1. Mapeamento de classes em Entity Framework Core.
2. Configuração de relacionamentos entre classes no Entity Framework Core.
3. Uso da anotação `[Key]` para especificar chaves primárias.
4. Configuração de herança em classes mapeadas no Entity Framework Core.
5. Resolução de problemas com a documentação do Swagger.
6. Remoção da necessidade de fornecer dados aninhados ao criar objetos.

Controllers e Depuração

Durante a construção dos controllers, o chat auxiliou fornecendo exemplos dos métodos além de revisar alguns conceitos básicos como a diferença entre Put/Patch. Neste processo também foi solicitado à IA que fizesse um método para validar CPF e CNPJ para cadastrar clientes, funcionários e fornecedores.

Durante a depuração o chat foi essencial, enviávamos os logs de erro do swagger pra ele e ele rapidamente entendia o erro e sugeria uma correção, coisas como routs com o mesmo nome ou nomes parecidos.

Algumas vezes foi necessário enviar partes do nosso código pra facilitar na identificação do problema como neste exemplo:



```
// Postar Vestimenta:
[HttpPost]
[Route("CadastrarVestimenta")]
public IActionResult CadastrarVestimenta(Vestimenta vestimenta)
{
    if (_dbContext is null) return NotFound();
    if (_dbContext.Vestimenta is null) return NotFound();


    _dbContext.AddAsync(vestimenta);
    _dbContext.SaveChangesAsync();
    return Created("", vestimenta);
}
```

assim está o método



O problema em seu código está relacionado ao uso incorreto dos métodos assíncronos `AddAsync` e `SaveChangesAsync`. Esses métodos são assíncronos e retornam uma `Task`, mas em seu código, você não está aguardando a conclusão dessas tarefas. Como resultado, o método `SaveChangesAsync` pode não ser chamado antes que a ação retorne.

Você deve usar a palavra-chave `await` para aguardar a conclusão dessas tarefas. Aqui está a correção do seu código:

 Regenerate