INJEÇÃO DE DEPENDÊNCIA E MIDDLEWARE EM RUBY ON RAILS (API MODE)

Oswaldo Beltrani Neto
2485990

SUMÁRIO

3. Introdução

4. Objetivo

5. Obstáculos

6. Arquitetura em camadas

7. Inversão de dependências

8. Middleware

9. Benefícios

10. Conclusão

INTRODUÇÃO

Ruby on Rails, ou simplesmente Rails, é um framework web de código aberto escrito em Ruby. Ele foi criado para agilizar o desenvolvimento de aplicações web, seguindo o princípio da convenção sobre configuração, ou seja, o desenvolvedor precisa escrever menos código porque o framework já assume padrões inteligentes.

Rails é baseado no padrão de arquitetura MVC (Model-View-Controller) e vem com uma série de ferramentas integradas que facilitam as Criação de APIs e aplicações completas, conexões com banco de dados, criações de rotas e validações.

OBJETIVO

Aplicar os conceitos de Inversão de Dependência (via injeção) e Middleware para tratamento de exceções no desenvolvimento de uma API modular, seguindo boas práticas de arquitetura em camadas.

OBSTÁCULOS

- Baixa compatibilidade com o sistema operacional Windows, não sendo o ambiente ideal...
  - Problemas de compatibilidade com gems nativas (que dependem de extensões C);
  - Sistema de arquivos mais lento para operações de compilação;
  - Definitivamente foi feito para ser usado via Linux, visto que, uma das soluções é simular um ambiente Linux para a sua utilização;
  - Após configurar o ambiente, a implementação foi super rápida.

ARQUITETURA EM CAMADAS

- Controller: Responsável por receber requisições e retornar respostas
- Service: Regras de negócio
- Repository: Comunicação com o banco de dados
- Middleware: Intercepta requisições e respostas (trata exceções)

Request → Controller → Service → Repository → DataBase

INVERSÃO DE DEPENDÊNCIA

Ao invés do controller criar diretamente a dependência, ele recebe uma instância de um serviço que, por sua vez, depende de um repositório. Isso segue o princípio de inversão de dependência (SOLID).

```ruby
def initialize
  super
  @service = ReservaService.new(ReservaRepository.new)
end
```

MIDDLEWARE

Responsável pelo tratamento centralizado de erros

- Problema: Tratamento de exceções dispersos por toda a aplicação;
- Solução: Um middleware que captura qualquer erro inesperado e retorna uma resposta padronizada.
- Exemplo: Middlewares/error_handler.rb

```
def call(env)
  begin
    @app.call(env)
  rescue => e
    error_response(e)
  end
end
```

Como ativamos o Middleware?

Basta registrar-lo em config/application.rg (equivalente a program.cs em C#)

Config.middleware.use ErrorHandler

BENEFÍCIOS

- Reuso e desacoplamento com injeção de dependência
- Legibilidade e organização com arquitetura em camadas
- Manutenção com tratamento centralizaado de erros

- Junção de todos: Código mais testável e escalável!

CONCLUSÃO

A aplicação dos conceitos de Injeção de Dependência e Middleware torna a arquitetura de uma API mais modular, escalável e fácil de manter. Com a injeção de dependência, conseguimos desacoplar componentes, facilitando testes e futuras mudanças na lógica de negócio. Já o uso de middleware permite centralizar o tratamento de erros e outras responsabilidades transversais, melhorando a confiabilidade e a experiência do desenvolvedor.

Essas práticas seguem princípios sólidos de engenharia de software e são fundamentais para construir APIs mais profissionais e confiáveis com Ruby on Rails.