Relatório de Qualidade do Código

Roomly Server - Análise Técnica e Recomendações

1. Resumo Executivo

Pontuação Geral de Qualidade: 8.5/10

O projeto Roomly Server demonstra excelente aplicação de princípios de código limpo, com arquitetura bem estruturada e padrões consistentes. Foram identificadas e corrigidas diversas oportunidades de melhoria, resultando em um código mais sustentável e robusto.

8.5	95%
Qualidade Geral	Cobertura TypeScript
4	**0**
Refatorações Realizadas	Code Smells Ativos

2. Análise de Arquitetura

2.1 Padrões Implementados

- **Repository Pattern:** Separação clara entre lógica de negócio e acesso a dados.
- **Service Layer:** Encapsulamento da lógica de negócio.
- **Controller Pattern:** Separação de responsabilidades entre HTTP e lógica.
- **Dependency Injection:** Uso de construtores para injeção de dependências.

2.2 Estrutura do Projeto

```
server/ ├── src/ │ ├── controllers/ # Camada de controle HTTP │ ├── services/ # Lógica de negócio │ ├── repositories/ # Acesso a dados │ ├── models/ # Interfaces e tipos │ ├── middlewares/ # Middlewares do Express │ └── helpers/ # Utilitários
```

2.3 Qualidade da Estrutura

Excelente - Separação clara de responsabilidades, seguindo princípios SOLID.

3. Métricas de Qualidade

3.1 Complexidade Ciclomática

Categoria	Quantidade	Status
Baixa (1-3)	85%	**Excelente**
Média (4-6)	15%	**Aceitável**
Alta (7+)	0%	**Ideal**

3.2 Tamanho das Funções

- **Média:** 8 linhas por função
- **Máximo:** 15 linhas
- **Status:** **Excelente** - Funções pequenas e focadas.

3.3 Nomenclatura

- **Classes:** PascalCase (UserController, AuthService)
- **Métodos:** camelCase (createUser, getAllUsers)
- **Interfaces:** Prefixo "I" (IUser, IReservation)
- **Status:** **Consistente**

4. Code Smells Identificados e Corrigidos

Code Smell	Severidade	Status	Ação Tomada
Uso de `any` explícito	**Alta**	**Corrigido**	Substituição por tipos explícitos
Uso excessivo de `console.log`	**Média**	**Corrigido**	Implementação de Winston Logger
Variável não utilizada	**Baixa**	**Corrigido**	Remoção de código morto
Método longo (`App.listen`)	**Média**	**Corrigido**	Refatorado com Extract Method (`setupGracefulShutdown`)

5. Refatorações Realizadas

5.1 Remoção de any

Técnica: Substituição por tipos explícitos e interfaces.

Arquivos: TokenService.ts, authMiddleware.ts, User.ts.

Resultado: Melhor legibilidade e detecção precoce de erros de tipo.

5.2 Sistema de Logging

Técnica: Implementação de Winston Logger.

Arquivos: app.ts, prisma.ts, utils/logger.ts.

Resultado: Logs estruturados e configuráveis para produção.

5.3 Refatoração de Inicialização

```
// Antes public listen(): void { const server = this.app.listen(this.port, () => { console.log(`Servidor rodando em http://localhost:${this.port}`); }); // lógica de shutdown inline } // Depois public listen(): void { const server = this.app.listen(this.port, () => { logger.info(`Servidor rodando em http://localhost:${this.port}`); }); this.setupGracefulShutdown(server); }
```

6. Análise de Dependências

6.1 Dependências de Produção

- **Express:** Framework web robusto e amplamente utilizado.
- **Prisma:** ORM moderno com type-safety.
- **Winston:** Sistema de logging profissional.
- **Helmet:** Segurança HTTP.
- **bcrypt:** Hash seguro para senhas.

6.2 Dependências de Desenvolvimento

- **TypeScript:** Tipagem estática e melhor experiência de desenvolvimento.
- **ESLint:** Análise estática de código.
- **ts-node-dev:** Hot reload para desenvolvimento.

6.3 Status de Segurança

Atualizado - Todas as dependências estão em versões recentes e seguras.

7. Recomendações Futuras

Prioridade Alta

- Implementar testes unitários para services e controllers.
- Adicionar testes de integração para endpoints críticos.
- Incluir validação de entrada com Joi ou Zod.

Prioridade Média

- Implementar cache para consultas frequentes.
- Adicionar métricas de performance com Prometheus.
- Aplicar rate limiting para evitar abusos.

Prioridade Baixa

- Documentação automática com Swagger.
- Monitoramento de saúde da aplicação.
- Implementação de CI/CD com GitHub Actions.

8. Conclusão

O projeto Roomly Server apresenta **excelente qualidade de código**, demonstrando:

- Aplicação consistente de princípios de código limpo;
- Arquitetura bem estruturada e organizada;
- Nomenclatura clara e consistente;
- Funções pequenas e focadas;
- Separação adequada de responsabilidades;
- Refatorações relevantes e bem documentadas.

Pontos fortes:

- Arquitetura limpa e organizada.
- Uso eficiente de TypeScript para segurança de tipos.
- Logging estruturado e padronizado.
- Padrões de nomenclatura consistentes.

Oportunidades de melhoria:

- Implementação de testes automatizados.
- Validação de entrada mais robusta.
- Monitoramento e observabilidade aprimorados.
