# 📊 RELATÓRIO DE ANÁLISE CONSOLIDADO - APIs de Engenharia de Software 2

**Data do Relatório:** 1 de dezembro de 2025
**Equipes Avaliadas:** Ana, Ruan e Willian
**Total de Testes Executados:** 160 (54 + 54 + 52)

---

## 📋 Sumário Executivo

Este relatório consolida a análise de três APIs desenvolvidas por diferentes equipes como parte do projeto de Engenharia de Software 2. Cada API foi submetida a uma bateria extensiva de testes automatizados cobrindo funcionalidades, validações, segurança e edge cases.

### Resultados Gerais Consolidados

| Equipe | URL da API | Tecnologia | Testes | Aprovados | Taxa | Nota Final |
|---|---|---|---|---|---|---|
| Ana | https://api-cores-node-bu6d.onrender.com | Node.js/Express | 52 | 30 | 57.7% | 5.5/10 🟠 |
| Ruan | http://atividadeengenharia2.infinityfree.me | PHP | 54 | 44 | 81.5% | 6.2/10 🟡 |
| Willian | https://cct.uenp.edu.br/coleti/es2/willian | PHP | 54 | 48 | 88.9% | 7.5/10 🟠 |

### 🏆 Classificação Final

1. 🥇 **Willian** - 7.5/10 - Boa base técnica, problemas críticos de performance
2. 🥈 **Ruan** - 6.2/10 - Funcional mas com falhas críticas de segurança
3. 🥉 **Ana** - 5.5/10 - Algoritmos excelentes, problemas de deploy

---

## 🎯 Análise Comparativa Detalhada

### 1. API de Análise e Paletas de Cores (Ana)

**Tecnologia:** Node.js + Express.js
**Hosting:** Render.com
**Funcionalidades:** Conversão de cores e geração de paletas

#### ☑ Pontos Fortes

1. **Algoritmos Complexos Implementados Manualmente**
   - Conversão HEX ↔ RGB sem bibliotecas externas
   - Conversão RGB ↔ HSL manual para paletas triádicas
   - Cálculo correto de cores complementares

2. **Código Limpo e Bem Estruturado**
    - Funções documentadas com JSDoc
    - Separação clara de responsabilidades
    - Código legível e organizado
3. **Documentação Integrada**
    - Endpoint `/docs` com informações completas
    - URLs de exemplo dinâmicas
    - Redirecionamento automático da raiz
4. **Performance Aceitável**
    - Tempo médio: ~1.88s por requisição
    - Responde dentro do timeout de 10s

### ❌ Problemas Críticos

1. 🟠 **Validação Inconsistente de Parâmetros**
    - Middleware valida apenas quando parâmetro existe
    - Parâmetros ausentes retornam 200 ao invés de 400
    - 5 testes de validação falharam
2. 🟠 **Erro 500 com Parâmetros Duplicados (REGRESSÃO)**
    - Correção documentada não está ativa em produção
    - `?hex=FF0000&hex=00FF00` causa crash do servidor
    - Evidência de problema de deploy
3. 🟠 **Header CORS Ausente**
    - Código tem `app.use(cors())` mas header não aparece
    - Pode ser problema do Render.com
    - API pode não funcionar em browsers
4. 🟡 **Tratamento de Rotas Inválidas**
    - Rotas inexistentes retornam 200 ao invés de 404
    - Violação de padrões REST

### 📊 Resultados por Endpoint

| Endpoint | Testes | Aprovados | Taxa |
|---|---|---|---|
| /hex_para_rgb | 5 | 3 | 60% |
| /calcular_complementar | 4 | 3 | 75% |
| /gerar_paleta_triadica | 4 | 3 | 75% |
| /obter_nome_cor | 5 | 3 | 60% |
| /docs | 2 | 2 | 100% |

### 🔍 Análise Crítica

**Situação Real:** Problema de Deploy vs Código

- 📝 **Código Fonte:** 8.0/10 - Correções documentadas em RELATORIO_ERROS.md
- 🌐 **Deploy (Render):** 5.0/10 - Versão desatualizada ou dependências faltando
- 📊 **Média Ponderada:** 6.5/10

**Evidências:**

- ☑ Trailing Slash/Case Sensitivity funcionam → Código base atualizado
- ❌ Parameter Pollution falha → Correção específica não deployada
- ❌ CORS ausente → Dependência `cors` não instalada no servidor

**Recomendação:** Fazer **redeploy completo** no Render.com. Com deploy correto, a pontuação subiria para **7.5-8.0/10**.

---

## 2. API de Validação (Ruan)

**Tecnologia:** PHP
**Hosting:** InfinityFree (Free Hosting)
**Funcionalidades:** Validação de e-mail, telefone, CPF e números positivos

### ☑ Pontos Fortes

1. **Validação de E-mail Impecável**
    - 100% de acertos (9/9 testes)
    - Aceita formatos complexos (subdomínios, caracteres especiais)
    - Rejeita corretamente e-mails malformados
2. **Validação de Números Perfeita**
    - 100% de acertos (8/8 testes)
    - Aceita inteiros e decimais
    - Rejeita corretamente negativos e zero
3. **Mensagens de Erro Claras**
    - Retorna mensagens descritivas
    - Lista de ações disponíveis quando inválidas
4. **Performance Excelente**
    - Tempo médio: ~300ms por requisição
    - Melhor performance entre as 3 APIs
5. **Segurança Contra Ataques Comuns**
    - SQL Injection bloqueado ☑
    - XSS bloqueado ☑
    - Unicode/Emojis tratados ☑

### ❌ Problemas Críticos

1. 🟤 **Validação de CPF Aceita Dígitos Repetidos**
   - CPFs como `11111111111`, `00000000000` são aceitos como válidos
   - Violação das regras da Receita Federal
   - Permite cadastros fraudulentos
   - **IMPACTO:** Falha de segurança grave
2. 🟤 **Formato de Resposta Não-Padrão**
   - Retorna HTML com JSON embutido em `<pre>`
   - Não retorna JSON puro
   - Content-Type: `text/html` ao invés de `application/json`
   - Dificulta integração com clientes HTTP padrão
3. 🟤 **Validação de Telefone Muito Restritiva**
   - Aceita apenas exatamente 9 dígitos
   - Rejeita formatos válidos brasileiros:
     - 11 dígitos (DDD + celular)
     - 10 dígitos (DDD + fixo)
     - 8 dígitos (fixo sem DDD)
4. 🟤 **API é Case-Sensitive**
   - `action=validar_email` funciona
   - `action=VALIDAR_EMAIL` retorna erro
   - UX ruim e inconsistente

### 📊 Resultados por Categoria

| Categoria | Testes | Aprovados | Taxa |
|---|---|---|---|
| Validação de E-mail | 9 | 9 | 100% |
| Validação de Tel. | 9 | 6 | 66.7% |
| Validação de CPF | 9 | 7 | 77.8% |
| Número Positivo | 8 | 8 | 100% |
| Tratamento de Erros | 3 | 3 | 100% |
| Segurança | 14 | 10 | 71.4% |

### 🔍 Análise Crítica

**Bugs Críticos de CPF Encontrados:**

- 🟤 `11111111111` - Aceito (deveria ser inválido)
- 🟤 `00000000000` - Aceito (deveria ser inválido)
- 🟤 `22222222222` - Aceito (deveria ser inválido)
- 🟤 `99999999999` - Aceito (deveria ser inválido)

**Recomendação:** ⚠ **NÃO USAR EM PRODUÇÃO** até corrigir:

1. Validação de CPF (adicionar verificação de dígitos repetidos)
2. Formato de resposta (retornar JSON puro)

---

## 3. API RESTful (Willian)

**Tecnologia:** PHP
**Hosting:** UENP (Servidor Acadêmico)
**Funcionalidades:** Cálculo de IMC, verificação de palíndromo, geração de tabuada

### ☑ Pontos Fortes

1. **Validações Sólidas**
    - Valores zero rejeitados corretamente
    - Valores negativos validados apropriadamente
    - Strings inválidas detectadas e rejeitadas
    - NaN e Infinity tratados adequadamente
2. **Segurança Robusta**
    - SQL Injection bloqueado ☑
    - XSS bloqueado (tags HTML removidas) ☑
    - Null Bytes tratados ☑
    - Unicode/Emojis tratados ☑
3. **Funcionalidades Avançadas**
    - Valores extremos suportados (números muito grandes)
    - Decimais aceitos e processados corretamente
    - Notação científica funciona perfeitamente
    - Textos longos processados (2001 caracteres)
4. **Classificações IMC Corretas**
    - 6 classificações testadas: 100% corretas
    - Abaixo do peso, normal, sobrepeso, obesidade I/II/III
5. **Mensagens de Erro Consistentes**
    - Formato JSON padronizado
    - Mensagens descritivas e específicas
    - Indicam exatamente qual parâmetro falta/é inválido

### ❌ Problemas Críticos

1. 🟤 **POST com JSON Não Funciona**
    - Documentação menciona suporte a POST + JSON
    - Todas requisições POST retornam erro
    - Força uso de query strings até para dados sensíveis
    - **IMPACTO:** Limita severamente usabilidade
2. 🟤 **Performance Muito Lenta**

- 5 requisições levaram 8.44 segundos
- Tempo médio: **1.69s por requisição**
- Inaceitável para API em produção
- Pode indicar problema de rede, DB ou código ineficiente

3. 🟡 **Métodos HTTP Não Documentados Aceitos**
   - PUT e DELETE retornam 200 (sucesso)
   - Header indica apenas GET e POST
   - Inconsistência entre documentação e implementação
   - Possível superfície de ataque não planejada
4. 🟡 **API é Case-Sensitive**
   - `acao=calcular_imc` funciona
   - `acao=CALCULAR_IMC` retorna erro
   - APIs modernas geralmente são case-insensitive
5. 🟡 **Não Remove Espaços em Branco**
   - `acao=" calcular_imc "` retorna erro
   - Sem aplicação de `trim()` nos parâmetros

### 📊 Resultados por Endpoint

| Endpoint | Testes | Aprovados | Taxa |
|---|---|---|---|
| calcular_imc | 14 | 13 | 93% |
| verificar_palindromo | 12 | 11 | 92% |
| gerar_tabuada | 10 | 9 | 90% |
| info_sistema | 1 | 1 | 100% |
| Testes Gerais | 17 | 12 | 71% |

### 📊 Resultados por Categoria

| Categoria | Cobertura | Status |
|---|---|---|
| Funcionalidades Básicas | 100% | ☑ |
| Validações | 100% | ☑ |
| Segurança | 100% | ☑ |
| Edge Cases | 100% | ☑ |
| Performance | 100% | ❌ |
| Compatibilidade | 67% | ❌ |

### 🔍 Análise Crítica

**Recomendações Prioritárias:**

**Alta Prioridade:**

1. Implementar suporte a POST com JSON
2. Investigar e otimizar performance (1.7s → <500ms)
3. Adicionar validação de métodos HTTP (retornar 405)

**Média Prioridade:** 4. Normalizar ações para lowercase 5. Aplicar trim em parâmetros 6. Validar parâmetros duplicados

---

## 📊 Comparação Lado a Lado

### Performance

| Equipe | Tempo Médio/Req | Status |
|---|---|---|
| Ruan | ~300ms | 🟢 Excelente |
| Ana | ~1.88s | 🟡 Aceitável |
| Willian | ~1.69s | 🔴 Inaceitável |

**Vencedor:** Ruan (5.6x mais rápido que Willian)

---

### Segurança

| Aspecto | Ana | Ruan | Willian |
|---|---|---|---|
| SQL Injection | ☑ | ☑ | ☑ |
| XSS | ☑ | ☑ | ☑ |
| CORS | ❌ | N/A | ☑ |
| Validação CPF | N/A | ❌ | N/A |
| Null Bytes | ☑ | ☑ | ☑ |
| Unicode | ☑ | ☑ | ☑ |

**Pontuação:**

- Ana: 4/5 (80%)
- Ruan: 4/6 (67%) - Penalizado pelo CPF
- Willian: 5/5 (100%)

**Vencedor:** Willian

---

### Formato de Resposta

| Equipe | Content-Type | Formato | Padrão REST |
|---|---|---|---|
| Ana | application/json | JSON | ☑ Sim |
| Ruan | text/html | HTML+JSON | ❌ Não |

| Equipe | Content-Type | Formato | Padrão REST |
|---|---|---|---|
| Willian | application/json | JSON | ☑ Sim |

**Vencedor:** Ana e Willian

---

## Tratamento de Erros

| Equipe | Mensagens Claras | HTTP Codes | Específico |
|---|---|---|---|
| Ana | ☑ | ⚠ | ☑ |
| Ruan | ☑ | ☑ | ☑ |
| Willian | ☑ | ☑ | ☑ |

**Vencedor:** Ruan e Willian

---

## Validações

| Tipo de Validação | Ana | Ruan | Willian |
|---|---|---|---|
| E-mail | N/A | 100% | N/A |
| Telefone | N/A | 67% | N/A |
| CPF | N/A | 78% | N/A |
| Cores HEX | 60% | N/A | N/A |
| Números Positivos | N/A | 100% | 93% |
| Strings | N/A | N/A | 100% |

**Análise:**

- Ruan: Excelente em e-mail (100%) mas falha crítica em CPF
- Ana: Boa em cores mas precisa melhorar validação de entrada
- Willian: Consistente em todas as validações

**Vencedor:** Willian (mais consistente)

---

## Case Sensitivity

| Equipe | Routes/Actions | Parâmetros | Status |
|---|---|---|---|
| Ana | ☑ Insensitive | ✗ Sem Trim | ◍ |
| Ruan | ✗ Sensitive | ✗ Sem Trim | ◍ |
| Willian | ✗ Sensitive | ✗ Sem Trim | ◍ |

**Observação:** Apenas Ana normaliza rotas (lowercase), mas nenhuma API faz trim de parâmetros.

**Vencedor:** Ana (parcial)

### Documentação

| Equipe | Documentação Integrada | Formato | Qualidade |
|---|---|---|---|
| Ana | ☑ Endpoint /docs | JSON | Excelente |
| Ruan | ❌ Não tem | N/A | N/A |
| Willian | ⚠ Parcial | N/A | Regular |

**Vencedor:** Ana

---

## 🎓 Conclusões e Recomendações Finais

### Ana - API de Cores

**Nota Final:** 5.5/10 → **Potencial de 7.5-8.0/10**

**Resumo:**

- ☑ Algoritmos complexos excelentemente implementados
- ☑ Código limpo e bem documentado
- ❌ Problemas críticos de deploy (não de código)

**Ações Imediatas:**

1. 🔥 Fazer redeploy completo no Render.com
2. 🔥 Executar `npm install` para instalar dependências (cors)
3. 🔥 Verificar logs do servidor para erros de inicialização
4. ⚡ Corrigir middleware de validação (parâmetros ausentes)
5. ⚡ Corrigir tratamento de rotas inválidas (retornar 404)

**Potencial:** Com correção do deploy, esta API tem potencial para **nota 8/10**.

---

### Ruan - API de Validação

**Nota Final:** 6.2/10

**Resumo:**

- ☑ Validações de e-mail e números perfeitas
- ☑ Performance excelente (300ms)
- ❌ Falha crítica de segurança no CPF
- ❌ Formato de resposta não-padrão

**Ações Imediatas:**

1. 🔥 **URGENTE:** Corrigir validação de CPF (rejeitar dígitos repetidos)
2. 🔥 **URGENTE:** Corrigir formato de resposta (retornar JSON puro)
3. ⚡ Melhorar validação de telefone (aceitar 8, 9, 10, 11 dígitos)
4. ⚡ Normalizar actions para lowercase
5. 💡 Aceitar CPF formatado (remover pontos/hífens)

**Código Sugerido para CPF:**

```php
function validarCPF($cpf) {
    $cpf = preg_replace('/[^0-9]/', '', $cpf);

    if (strlen($cpf) != 11) return false;

    // ADICIONAR: Rejeitar dígitos repetidos
    if (preg_match('/(\d)\1{10}/', $cpf)) {
        return false;
    }

    // ... resto da validação
}
```

**Recomendação:** ⚠ **NÃO USAR EM PRODUÇÃO** até corrigir os 2 problemas críticos.

---

## Willian - API RESTful

**Nota Final:** 7.5/10

**Resumo:**

- ☑ Validações sólidas e segurança robusta
- ☑ Funcionalidades avançadas funcionando
- ❌ Performance inaceitável (1.7s/req)
- ❌ POST com JSON não funciona

**Ações Imediatas:**

1. 🔥 **CRÍTICO:** Implementar suporte a POST com JSON
2. 🔥 **CRÍTICO:** Investigar e otimizar performance
   - Verificar latência de rede
   - Analisar logs do Apache
   - Otimizar código PHP (cache, queries)
3. ⚡ Adicionar validação de métodos HTTP (405)
4. ⚡ Normalizar ações para lowercase
5. ⚡ Aplicar trim em parâmetros

**Código Sugerido para POST JSON:**

```php
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $contentType = $_SERVER['CONTENT_TYPE'] ?? '';

    if (strpos($contentType, 'application/json') !== false) {
        $json = file_get_contents('php://input');
        $_POST = json_decode($json, true) ?? [];
    }
}
```

**Recomendação:** Após correção dos 2 problemas críticos, a API está pronta para produção.

---

## Ranking Final

### Por Nota Geral

1. 🥇 **Willian** - 7.5/10
2. 🥈 **Ruan** - 6.2/10
3. 🥉 **Ana** - 5.5/10

### Por Potencial (com correções)

1. 🥇 **Ana** - 8.0/10 (após deploy correto)
2. 🥈 **Willian** - 8.5/10 (após POST JSON + performance)
3. 🥉 **Ruan** - 7.5/10 (após CPF + formato resposta)

### Por Performance

1. 🥇 **Ruan** - 300ms ⚡
2. 🥈 **Willian** - 1.69s 🚗
3. 🥉 **Ana** - 1.88s 🚗

### Por Segurança

1. 🥇 **Willian** - 100%
2. 🥈 **Ana** - 80%
3. 🥉 **Ruan** - 67% (penalizado pelo CPF)

### Por Taxa de Sucesso

1. 🥇 **Willian** - 88.9%
2. 🥈 **Ruan** - 81.5%
3. 🥉 **Ana** - 57.7%

---

## Estatísticas Consolidadas

### Totais Gerais

- **APIs Analisadas:** 3
- **Testes Executados:** 160
- **Testes Aprovados:** 122 (76.3%)
- **Testes Falhados:** 38 (23.7%)
- **Problemas Críticos:** 7
- **Problemas Médios:** 11
- **Observações:** 6

### Problemas Críticos por Categoria

- **Deploy/Configuração:** 2 (Ana: CORS, Parameter Pollution)
- **Performance:** 1 (Willian: 1.7s/req)
- **Segurança:** 1 (Ruan: CPF com dígitos repetidos)

- **Formato/Padrões:** 2 (Ruan: HTML+JSON, Willian: POST não funciona)
- **Validação:** 1 (Ana: parâmetros ausentes)

### Pontos Fortes Comuns

- ☑ Segurança contra SQL Injection (3/3)
- ☑ Segurança contra XSS (3/3)
- ☑ Tratamento de Unicode/Emojis (3/3)
- ☑ Mensagens de erro claras (3/3)

### Pontos Fracos Comuns

- ❌ Case-sensitivity desnecessária (3/3)
- ❌ Sem trim de parâmetros (3/3)
- ❌ Sem rate limiting (3/3)
- ❌ Sem testes unitários integrados (3/3)

---

## 🎯 Recomendações Gerais para Todas as APIs

### Alta Prioridade

1. **Implementar Rate Limiting**
   - Proteger contra ataques de força bruta/DDoS
   - Sugestão: 100 requisições/minuto por IP
2. **Normalizar Inputs**
   - Aplicar `strtolower()` em actions/endpoints
   - Aplicar `trim()` em todos os parâmetros
3. **Adicionar Testes Unitários**
   - Jest/Mocha para Node.js (Ana)
   - PHPUnit para PHP (Ruan e Willian)
4. **Documentação Padronizada**
   - OpenAPI/Swagger
   - Exemplos de uso em múltiplas linguagens

### Média Prioridade

5. **Logging Estruturado**
   - Winston/Morgan (Node.js)
   - Monolog (PHP)
6. **Monitoramento**
   - Tempo de resposta
   - Taxa de erros
   - Uso de recursos

7. **Versionamento de API**
    - `/v1/endpoint` para futuras versões

### Baixa Prioridade

8. **Suporte a Múltiplos Formatos**
    - Accept: application/json, application/xml
9. **Cache de Respostas**
    - Redis para respostas frequentes
10. **Documentação de Limites**
    - Tamanho máximo de requisição
    - Timeout
    - Rate limits

---

## Conclusão Final

As três APIs demonstram competência técnica sólida com diferentes pontos fortes:

- **Ana** se destaca pela complexidade dos algoritmos e código limpo, mas sofre com problemas de deploy
- **Ruan** tem a melhor performance mas falha criticamente na validação de CPF e formato de resposta
- **Willian** apresenta a solução mais completa e equilibrada, mas precisa urgentemente resolver performance e suporte a POST JSON

**Nenhuma das APIs está pronta para produção no estado atual**, mas todas têm potencial para se tornarem soluções profissionais após as correções recomendadas.

Com as correções implementadas, o ranking potencial seria:

1. **Ana** - 8.0/10 (algoritmos complexos + correções)
2. **Willian** - 8.5/10 (base sólida + correções)
3. **Ruan** - 7.5/10 (performance excelente + correções)

---

**Relatório consolidado baseado em:**

- Ana: 52 testes automatizados (Python)
- Ruan: 54 testes automatizados (PowerShell)
- Willian: 54 testes automatizados (Python)

**Total:** 160 testes executados com cobertura completa de funcionalidades, validações, segurança e edge cases.