

Task DevOps

Workflow com Github Actions

QA Líder: Alyson Campos

Squad 7: Larissa Caldeira e Yuri Baruk

Sumário

Task DevOps.....	1
Sumário	2
Objetivo do projeto.....	3
Estruturação dos testes.....	4
Health Check.....	4
Contrato.....	4
Funcional	4
Exemplo de Teste após Reestruturação:	5
Ferramentas utilizadas	6
Arquitetura da pipeline	8
Resultados obtidos	9
Referências	12



Objetivo do projeto

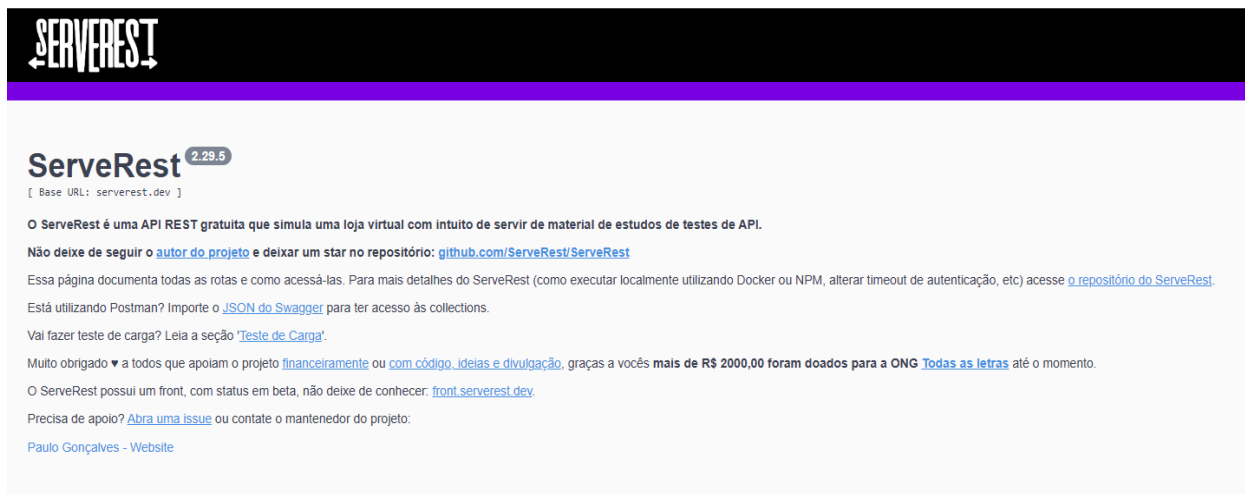
Cenário: Você é o responsável por garantir a qualidade de um sistema de e-commerce em constante evolução. Para agilizar o processo de testes e garantir a entrega contínua de valor, você precisa implementar uma pipeline de testes automatizados.

Tarefas:

- Crie um repositório no Github para o projeto de testes de API.
- Configure o ambiente de desenvolvimento local com as ferramentas escolhidas.
- Criação dos Testes: Crie testes automatizados de API para endpoints relevantes para a API [ServeRest](#).
- Integração com o Github Actions:
 - Crie um arquivo de workflow no GitHub Actions para definir a pipeline de testes.
 - Configure a pipeline para executar os testes de API.
 - Adicione etapas para executar os testes de Health check, Contrato e Funcional.
- Relatório de Testes: Configure a pipeline para gerar relatórios de testes com Allure Report e exibir os resultados no Github Pages.
- Plus: Integre a pipeline com a ferramenta de comunicação Discord para notificar os resultados dos testes.
- Plus: Integre o SonarQuBe



Objeto de Testes



Estruturação dos testes

Neste projeto, aproveitamos os testes realizados anteriormente na Task final de RestAssured separamos um total de 24 testes dentre eles os endpoints para testar as funcionalidades de login e usuário, porém os reorganizamos para se adequarem ao padrão da pipeline, categorizando-os da seguinte forma:

Health Check

Testes responsáveis por verificar a disponibilidade e o funcionamento básico da API, assegurando que ela está operacional e respondendo como esperado.

Contrato

Testes focados em validar a estrutura das respostas da API, garantindo que o formato do JSON está conforme o especificado e que todos os componentes esperados estão presentes.

Funcional

Testes que avaliam as funcionalidades da API, verificando se as respostas retornadas estão corretas e em conformidade com as regras de negócio estabelecidas.

Além dessas categorias, também adicionamos referências e anotações do **Allure Report** para organizar melhor o relatório, utilizando:



- **Display Name:** Para definir um título claro para os testes e para o switch de testes.
- **Severity:** Para classificar a gravidade dos testes, com níveis como “trivial”, “minor”, “normal”, “crítico” e “bloqueador”.
- **Epic, Feature, Story:** Para estruturar uma hierarquia baseada em comportamento, seguindo as boas práticas do Allure.

Exemplo de Teste após Reestruturação:

```
@Epic("API")
@Feature("FUNCIONALIDADES")
@Story("BUSCAR USUÁRIOS POR ID")
@DisplayName("Testes de buscar usuários por ID")
public class BuscarUsuariosPorIdTest {

    private final UsuarioClient usuarioClient = new UsuarioClient(); 5 usages

    @Test
    @DisplayName("CT003 - Validar buscar usuário por ID")
    @Tag("HealthCheck")
    @Severity(SeverityLevel.NORMAL)
    public void testBuscarUsuarioPorIdComSucesso() {

        UsuarioRequest usuario = UsuarioDataFactory.usuarioValido();

        String idUsuario = usuarioClient.cadastrarUsuarios(usuario) Response
        .then() ValidatableResponse
            .statusCode(i: 201)
            .extract().path(s: "_id")
        ;

        usuarioClient.buscarUsuariosPorId(idUsuario) Response
        .then() ValidatableResponse
            .statusCode(i: 200);
    }
}
```

- As referências e anotações foram obtidas na [Documentação do Allure](#).



Ferramentas utilizadas

Durante o projeto foram utilizadas as seguintes ferramentas:

- Allure Report:
 - Ferramenta utilizada para gerar relatório de todos os testes, indicando se passaram, deram falha, erro ou se foram ignorados por algum motivo. Além disso, a ferramenta também traz mais opções de uso que podem ser lidas melhor na [documentação oficial](#).
- Github Actions:
 - Ferramenta utilizada para entrega contínua e integração contínua (CI/CD) que permite automatizar a compilação, teste e implementação da pipeline. Mais informações podem ser obtidas na [documentação do Github Actions](#).
- SonarQuBe:
 - Ferramenta web, que pode ser integrada com vários serviços de criação de pipelines e acompanhamento de projeto, que permite verificar o código através do uso de IA, para detectar falhas e possíveis melhorias no código, evitando duplicação de métodos, exposição de dados e garantindo a segurança. Leia mais na [documentação oficial](#).
- RestAssured:
 - Ferramenta Java utilizada para criar e gerenciar testes de requisições de API Rest que pode utilizar extensões *Maven* ou *Gradle* para ajudar na criação de códigos limpos e bem estruturados. Leia mais na [documentação oficial](#).
- Java Development Kit:
 - Como nossos testes precisam ser escritos e executados na linguagem de programação Java, optamos por usar o kit de desenvolvimento java na versão 17, pois é a versão mais estável e segura, com compatibilidade a praticamente todas as ferramentas



extraoficiais que nos ajudam a escrever o projeto. Confira mais na [documentação oficial](#).

- Maven:
 - Ferramenta de gerenciamento e compreensão de projetos de software baseado no conceito de um modelo de objeto de projeto (POM), pode gerenciar a construção, o relatório e a documentação de um projeto a partir de uma informação central, leia mais sobre na [Documentação Maven](#).
- Discord - **WebHook**:
 - Ferramenta de comunicação muito utilizada atualmente, com a integração com o Github Actions é possível enviar notificações sobre a execução de pipelines diretamente em seu servidor. Para implementar essa funcionalidade, foi utilizada a integração disponibilizada pela página [Discord for GitHub Actions](#).



Arquitetura da pipeline

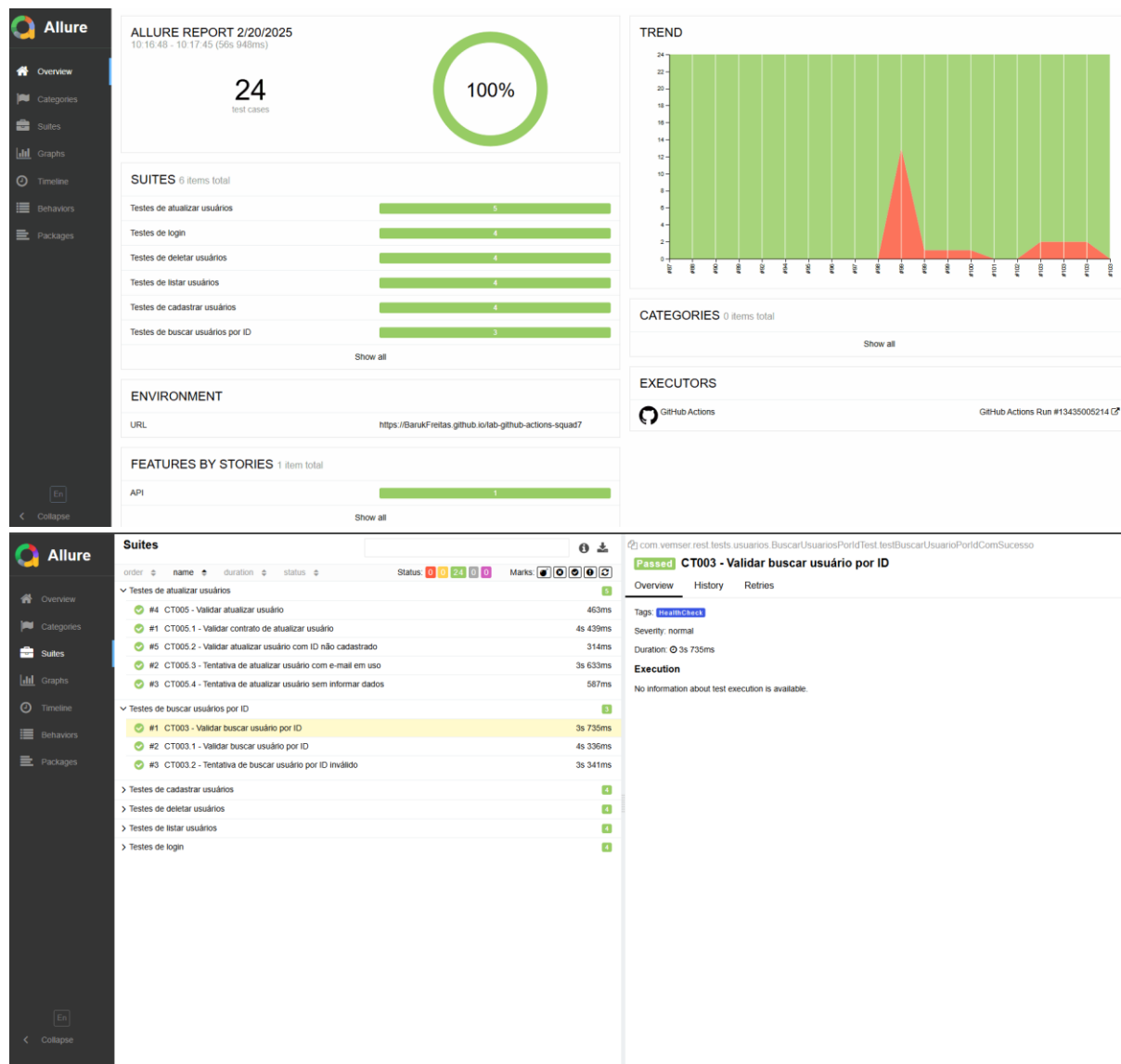


Nossa pipeline executa um script que irá compilar o código, rodar alguns testes para verificar se a API está operante, caso esse teste seja positivo, prosseguirá para os testes de contrato, que verifica se os testes programados estão seguindo o esquema JSON definido previamente na [documentação da API](#), caso esse teste seja positivo, prosseguirá novamente para a próxima etapa que irá testar todas as funcionalidades. Nesta etapa, ele irá fazer testes positivos e negativos, portanto, independente se ocorram erros, o relatório irá ser gerado para que assim possamos acompanhar o progresso dos testes, mostrando quais passaram, quais falharam, onde falharam. Além de tudo, o link gerado com o relatório será hospedado [neste link](#), que é atualizado sempre que a pipeline é executada, garantindo que o relatório seja sempre dos resultados coletados recentemente. Este link também é enviado automaticamente para um [servidor no discord](#), dedicado apenas para que possamos ver as atualizações feitas no projeto.



Resultados obtidos

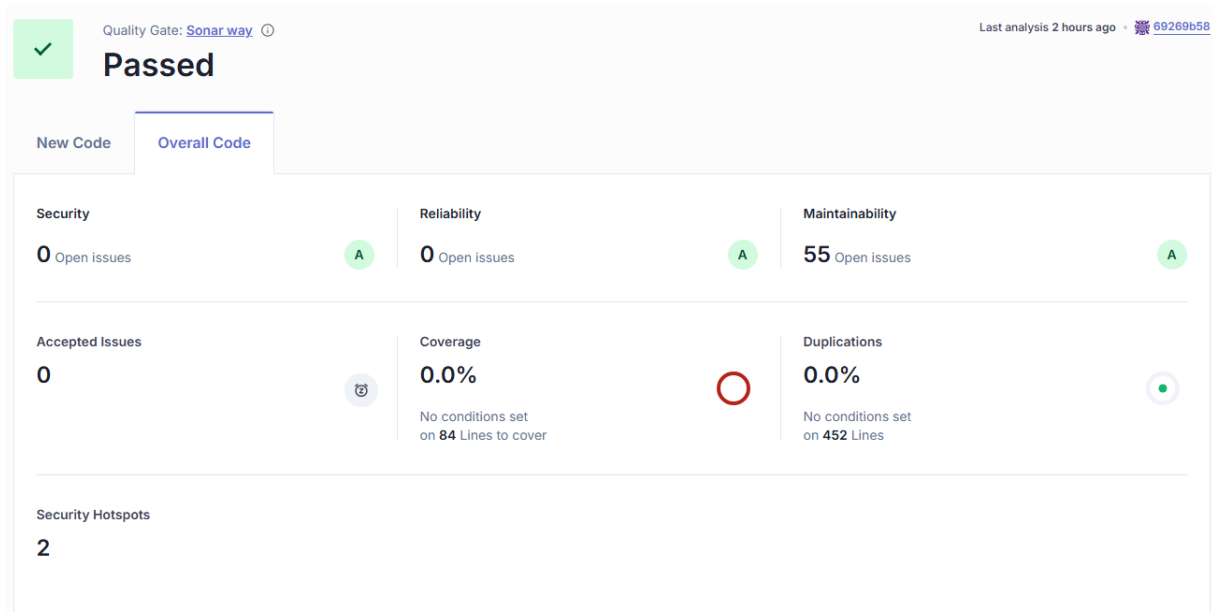
Após a implementação e execução da pipeline, o relatório é enviado diretamente para o nosso servidor do Discord. Utilizando o Allure Report, é possível acompanhar o registro detalhado de todos os testes realizados. Dessa forma, conseguimos monitorar facilmente a aparição de novos erros, bem como verificar se os erros identificados anteriormente foram corrigidos.



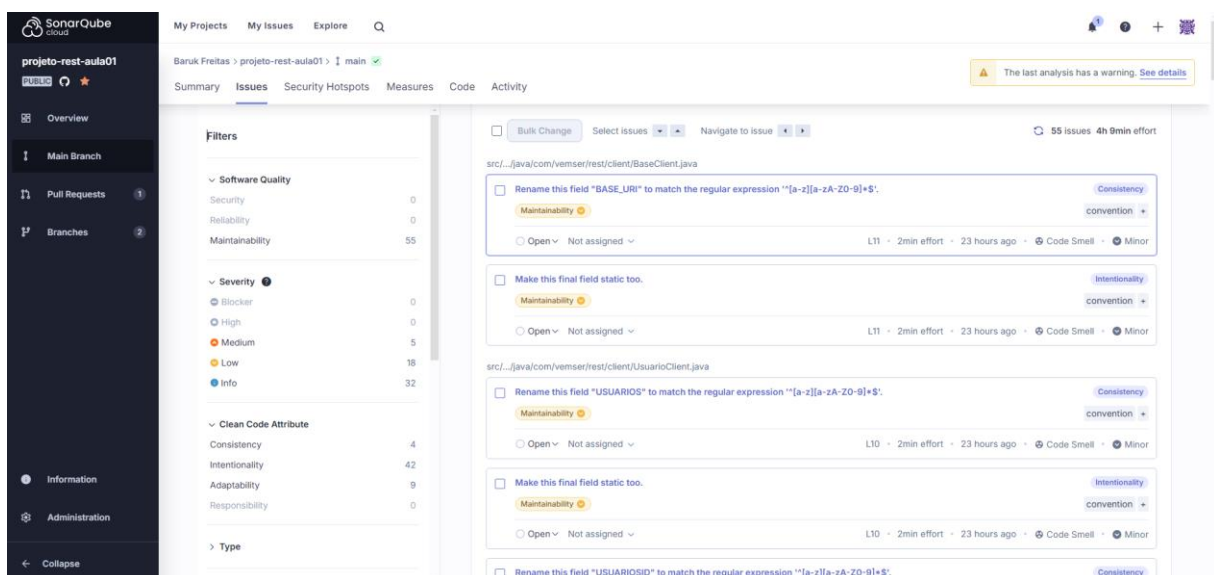
Durante todo o processo, obtivemos um resultado positivo: **todos os 24 testes foram aprovados**, sem a detecção de nenhum bug durante a fase de testes no **ServerRest**.



Depois da implementação do SonarQube, conseguimos notar como ele ajuda a pontuar onde os desenvolvedores que irão consertar a aplicação devem concentrar seus focos, pois ele mostra exatamente onde que tem códigos duplicados, onde que tem possíveis erros, onde devem existir manutenções, além de mostrar se a qualidade do código está aceitável ou não.



Nessa tela é possível observar todos os possíveis erros no código, classificados automaticamente com severidade de baixa a alta, blocker ou info.



Quando o impacto do erro é classificado em alto, significa por maioria das vezes, que tem variáveis não utilizadas ou linhas repetidas que podem ser simplificadas. Além disso, ele também mostra exatamente onde que está a variável e sugere a correção para ela.

Intentionality | Not clear

Remove this unused method parameter "usuario". [↗](#)

Unused method parameters should be removed [java:S1172](#)

Software qualities impacted: Maintainability

☐ Open ☐ Not assigned ☒ Code Smell ☒ Major

Tags

cert ... +

Line affected

L36

Effort

5 min

Introduced

23 hours ago

Where is the issue?

Why is this an issue?

How can I fix it?

Activity

More info

Open in IDE

35 -

36 -

37 -

38 -

39 -

40 -

41 -

42 -

43 -

44 -

45 -

```
public Response listarUsuarios(UsuarioRequest usuario){  
  
    return  
        given()  
            .spec(super.set())  
            .when()  
                .get(USUARIOS)  
            ;  
}
```



Referências

- ☐ [Documentação oficial Java](#)
- ☐ [Documentação Maven.](#)
- ☐ [Documentação oficial RestAssured](#)
- ☐ [Documentação do Allure.](#)
- ☐ [Documentação do Github Actions](#)
- ☐ [Discord for GitHub Actions.](#)
- ☐ [Documentação oficial SonarQube](#)
- ☐ [Usando Github Actions e Discord](#)
- ☐ [Como Usar os Segredos do GitHub Actions](#)

