UNIVERSIDADE FEDERAL DE UBERLÂNDIA

João Vitor Almeida Alves

Melhorias na Infraestrutura do Sistema de Cadastro de Atividades Docente

Uberlândia, Brasil

2022

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

João Vitor Almeida Alves

Melhorias na Infraestrutura do Sistema de Cadastro de Atividades Docente

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Sistemas de Informação.

Orientador: Rodrigo Sanches Miani

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Sistemas de Informação

Uberlândia, Brasil

2022

João Vitor Almeida Alves

Melhorias na Infraestrutura do Sistema de Cadastro de Atividades Docente

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Sistemas de Informação.

Trabalho aprovado. Uberlândia, Brasil, 29 de Março de 2022:

Rodrigo Sanches Miani
Orientador

Bruno Augusto Nassif Travençolo

Mauricio Cunha Escarpinati

Uberlândia, Brasil
2022

"Eu não tenho ídolos. Tenho admiração por trabalho, dedicação e competência."

Ayrton Senna da Silva

Resumo

Este trabalho apresenta melhorias implementadas no Sistema de Cadastro de Atividades Docente (SCAD). O SCAD foi desenvolvido em 2019 com o intuito de auxiliar o docente da Universidade Federal de Uberlândia no seu processo de progressão de carreira através da automatização da confecção dos relatórios e do cálculo da pontuação das atividades, ambos cruciais neste processo. Com o uso, foram identificadas necessidades de melhoria na segurança e no processo de gerenciamento e publicações de novas versões deste sistema, objetivos deste trabalho. O trabalho realizado automatizou todo o processo de gerenciamento e publicações do sistema, utilizando o Jenkins, o que mitigou possíveis falhas humanas neste processo que era manual, gerou indicadores a respeito da saúde do código fonte, com o SonarQube, e fortaleceu a segurança do sistema, ponto muito relevante para um sistema que mantém dados importantes.

Palavras-chave: Sistema de informação, Carreira, Progressão, DevOps, Docentes, Segurança da informação.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

SCAD	Sistema de Cadastro de Atividades Docente
UFU	Universidade Federal de Uberlândia
API	Interface de programação de aplicações (*Application Programming Interface*)
SEI	Sistema Eletrônico de Informações
HTTP	Protocolo de Transferência de Hipertexto (*Hyper Text Transfer Protocol*)
HTTPS	Protocolo de Transferência de Hipertexto Seguro (*Hyper Text Transfer Protocol Secure*)
SO	Sistema Operacional
ISO	Organização Internacional de Padronização (*International Organization for Standardization*)
RAM	Memória de acesso aleatório (*Random Access Memory*)
CPU	Unidade central de processamento (*Central Processing Unit*)
DNS	Sistema de nome de domínio (*Domain Name System*)
JVM	*Java Virtual Machine*
JRE	*Java Runtime Environment*
SGBD	Sistema de Gerenciamento de Banco de Dados
CORS	Compartilhamento de recursos com origens diferentes (*Cross-Origin Resource Sharing*)
URL	Localizador de recursos uniforme (*Uniform Resource Locator*)

Sumário

1 Introdução

Os docentes da Universidade Federal de Uberlândia (UFU), para viabilizar sua progressão de carreira, têm de produzir relatórios das atividades desenvolvidas em um determinado período. O trâmite dessa progressão se dá por meio do Sistema Eletrônico de Informações (SEI), que é o sistema responsável por gerir todos os processos da Universidade. Ainda que o processo tramite através de um ambiente eletrônico, identificou-se a necessidade de um sistema de informação capaz de padronizar a confecção dos relatórios e gerir os dados pertinentes a eles - esse sistema foi implementado e nomeado como Sistema de Cadastro de Atividades Docente, ou SCAD. O SCAD está em uso na Faculdade de Computação da UFU desde 2019, quando foi construído (SILVA, 2019), e apoia os docentes na medida em que automatiza o cálculo das pontuações das atividades, possibilita o *upload* e a manutenção dos documentos comprobatórios e gera, de forma automatizada, os relatórios que são inseridos no SEI.

Como consequência de sua relevância e uso contínuo entre os docentes da Faculdade de Computação (FACOM), foi possível identificar alguns pontos de melhorias, além de novas funcionalidades que foram implementadas no final de 2021 (REIS, 2021). Em tal trabalho foram reformulados os módulos de gerenciamento de atividades, o módulo de relatórios e o módulo de gerenciamento do avaliador. Outra necessidade decorrente do uso contínuo é a de garantir que esse uso seja seguro, bem como a de criar uma condição favorável à manutenção e ao desenvolvimento de novas funcionalidades apoiando-se em um processo de integração contínua.

1.1 Objetivo

O objetivo deste trabalho consiste em implementar ferramentas de suporte à manutenção do SCAD, no que diz respeito ao desenvolvimento ou correção de funcionalidades, processos e garantia da segurança do sistema.

1.1.1 Objetivos Específicos

- Criar um processo de integração e entrega contínua para o SCAD.
- Identificar pontos de melhoria de performance na base de código do SCAD.
- Identificar possíveis problemas de segurança na base de código e no servidor do SCAD.
- Alterar uma política insegura de *Cross Origin Resource Sharing*, ou CORS.

- Substituir as requisições do sistema que utilizam o protocolo *Hyper Text Transfer Protocol* (HTTP) pelo *Hyper Text Transfer Protocol Secure* (HTTPS).

1.2 Justificativa

Após as melhorias desenvolvidas por (REIS, 2021), identificou-se a necessidade de se realizar uma esteira de integração e entrega contínua, uma vez constatada uma grande complexidade durante o processo de atualização do sistema no servidor da UFU - processo que era susceptível à erros humanos, pois era feito de forma manual.

Para além do apontamento supracitado, é possível identificar uma tendência favorável, no mercado de desenvolvimento de *software*, à adoção de DevOps como cultura nas organizações (FORSGREN et al., 2019). Entre as justificativas para essa medida, estão a necessidade de garantir escalabilidade para os sistemas, a demanda por velocidade na entrega de *software*, a preocupação com a saúde mental dos profissionais que atuam com desenvolvimento de *software*, e a busca por sistemas cada vez mais seguros (FORSGREN et al., 2019). Logo, trazer esta perspectiva para os sistemas produzidos e utilizados no meio acadêmico, que não possuem os mesmos recursos do mercado privado, pode trazer os mesmos benefícios para esta comunidade.

Nesse sentido, um ponto extremamente crítico, que compromete a segurança de sistemas atualmente, é a utilização de requisições HTTP (*Hyper Text Transfer Protocol*), protocolo que já se provou inseguro e foi melhorado com a adoção de uma camada de criptografia no seu sucessor; o HTTPS (STALLINGS, 2006). O SCAD faz uso do protocolo HTTP, portanto, se faz necessária a atualização do mesmo para que os dados sejam trafegados via HTTPS. Essa atualização, por sua vez, tornará o sistema mais resistente do ponto de vista de segurança das informações nele trafegadas.

1.3 Organização do Trabalho

Este trabalho contém quatro capítulos. No Capítulo 1 trata-se da introdução, objetivos e justificativa. No Capítulo 2 são apresentadas as tecnologias e a arquitetura utilizada para desenvolver o SCAD e o processo de integração e entrega contínua. O Capítulo 3 abordará de forma explicativa como se deu o desenvolvimento e implantação do processo de integração e entrega contínua, aliado à correção das falhas de segurança relacionadas ao uso do protocolo HTTP. Por fim, no Capítulo 4 se encontra a conclusão e discussões sobre trabalhos futuros.

2 Referencial Teórico

Este capítulo apresenta a arquitetura escolhida, e as tecnologias utilizadas no desenvolvimento do sistema e no processo de integração e entrega contínua.

2.1 Arquitetura

Esta seção expõe as tecnologias que estiveram presentes no desenvolvimento do sistema, no processo de integração e entrega contínua e na implementação das correções no que diz respeito à segurança.

A solução foi construída utilizando o conceito de Interface de Programação de Aplicações (API) onde serviços são expostos através de uma determinada interface, e podem ser consumidos por diferentes clientes. No SCAD a API foi construída utilizando a linguagem C#, com o suporte do *framework* .NET Core, também chamamos esta faceta do sistema de *back-end*, nela são expostas rotas HTTP através das quais é possível manipular os dados armazenados pela aplicação, que são gerenciados pelas regras de negócio implementadas nesta. Já o *front-end*, ou o cliente da API, foi construído utilizando o *framework* Angular2+ que tem suas base de código escrita em Typescript. É relevante destacar que uma vez que esse modelo de arquitetura foi utilizado é possível que sejam agregados, futuramente, diferentes clientes, ou *front-ends* para a solução, como por exemplo uma versão para dispositivos móveis, sem que haja alteração no *back-end*.

O processo de integração e entrega contínua foi construído usando como plataforma o Jenkins, através do qual é possível agregar diferentes partes do processo, como por exemplo, o *download* do código fonte de um repositório do GitHub e a análise estática de código realizada pelo SonarQube.

2.2 Tecnologias

Abaixo estão listadas as principais ferramentas utilizadas para apoiar o processo de desenvolvimento:

- **C#**, C# *(C Sharp)* é uma linguagem de programação orientada a objeto, desenvolvida pela Microsoft, descendente da linguagem C e é utilizada na arquitetura .NET. (MICROSOFT, 2022a)
- **.NET Core**, .NET Core é uma plataforma *open-source* para desenvolvimento e execução de sistemas web, *desktop* e *mobile* desenvolvida pela Microsoft (MICROSOFT,

2022b).

- **Angular 2+**, Angular é um *framework* desenvolvido pela Google, sua principal aplicação é a construção de sistemas *front-end* (ANGULAR, 2022).
- **PostgreSQL**, o PostgreSQL é um sistema gerenciador de bancos de dados relacional de código aberto, com uma excelente reputação na comunidade por conta da sua confiabilidade e performance (POSTGRESQL, 2022).
- **VirtualBox**, o VirtualBox é uma solução de virtualização que tem suporte para diversos sistemas operacionais e distribuições (ORACLE, 2022).
- **Jenkins**, o Jenkins é um servidor de automação de código aberto, capaz de automatizar a maioria das tarefas de um ciclo de vida de *software*, como por exemplo *build*, *deploy*, testes de unidade e análise estática de código (JENKINS, 2022).
- **Groovy**, Groovy é uma linguagem de programação orientada a objetos que é compilada para *bytecode* Java e roda sobre a *Java Virtual Machine* (JVM) (GROOVY, 2022).
- **SonarQube**, o SonarQube é uma ferramenta de análise estática de código, com este é possível extrair métricas sobre *bugs*, *code smells* e possíveis *leaks* de segurança presentes no código (SONARQUBE, 2022a).
- **Webhook Relay**, o Webhook Relay é uma ferramenta que serve como um *proxy* de webhooks, que são chamadas de informação enviadas via HTTP para indicar algum comportamento, como por exemplo, a atualização de um repositório do GitHub. O Webhook Relay cria, localmente, uma interface para aguardar chamadas de webhook tirando a necessidade de expor o servidor, no caso o Jenkins, à internet (WEBHOOKRELAY, 2022).
- **Internet Information Services (IIS)**, o IIS é um servidor *web* desenvolvido pela Microsoft parar hospedar sites e conteúdos que se queira disponibilizar na *internet* (MICROSOFT, 2022c).
- **WebDeploy**, o WebDeploy é uma extensão do IIS, utilizada para gerenciar a implantação de sistemas no servidor do IIS (MICROSOFT, 2022d).
- **Mkcert**, mkcert é uma ferramenta disponibilizada no GitHub de forma aberta para criar localmente uma autoridade certificadora e um certificado válido e assinado por esta de forma a possibilitar testes que envolvam aplicações que vão futuramente ser assinadas por autoridades certificadoras reais (FILOSOTTILE, 2022).

3 Desenvolvimento

Este capítulo apresenta o processo de construção da esteira de integração e entrega contínua, todos os passos estruturantes do ponto de vista de ambiente de desenvolvimento, configurações dos *softwares* que deram suporte a esse processo, e por último melhorias de segurança que foram identificadas como consequência da própria implantação da esteira.

3.1 Etapas do Desenvolvimento

O desenvolvimento foi dividido em quatro grande etapas, que serão detalhadas à seguir, organizadas da seguinte maneira:

- **Preparação do ambiente de testes**, o objetivo desta etapa foi criar um servidor de testes com características análogas ao servidor que hoje abriga o SCAD, apresentado com detalhes na Seção 3.2.
- **Criação da esteira de integração contínua**, nesta etapa, apresentada na Seção 3.3, foi adicionado e configurado no servidor o *software* responsável pela automação da esteira de entrega e integração contínua.
- **Integração do SonarQube à esteira de integração contínua**, a finalidade deste passo era inserir uma ferramenta de análise estática do código que agregasse valor ao processo pois, além da automação a esteira, passou a entregar indicadores relevantes em termos de segurança e qualidade do código de forma contínua. Este passo foi detalhado na Seção 3.4.
- **Melhorias de segurança**, neste momento, detalhado na Seção 3.5, foram implementadas duas grandes melhorias de segurança na aplicação.

3.2 Preparação do Servidor de Testes

Uma vez que o ambiente, ou servidor de aplicação, real tem o acesso restrito pois fica em uma máquina física hospedada na FACOM, e que o sistema se encontra publicado e em funcionamento, para que os testes ocorressem foi necessária a criação de um ambiente fictício que simulasse as condições, denominado de ambiente de desenvolvimento daqui pra frente.

3.2.1 Criação da Máquina Virtual

A máquina onde se encontra o sistema tem como Sistema Operacional (SO) o Windows Server e a maneira escolhida para simular este cenário foi criando uma máquina virtual, para isso utilizou-se a ferramenta gratuita VirtualBox, que oferece suporte a virtualização de vários sistemas operacionais e suas distribuições.

Para emular uma máquina o VirtualBox precisa de uma imagem ISO do SO que se deseja e para a criação da máquina do ambiente de desenvolvimento foi utilizada uma ISO do Windows Server 2019, disponibilizada de forma oficial pela Microsoft com uma licença de avaliação gratuita de 180 dias (MICROSOFT, 2022e).

Além do sistema operacional, o VirtualBox permite que se configure, com a limitação do *hardware* da máquina hospedeira, aspectos do *hardware* da máquina virtual como a memória RAM, a quantidade de CPUs, memória de vídeo, e também o tamanho do disco de armazenamento. As configurações utilizadas na máquina do ambiente de desenvolvimento podem ser observadas na Figura 1, que é uma captura de tela do VirtualBox.

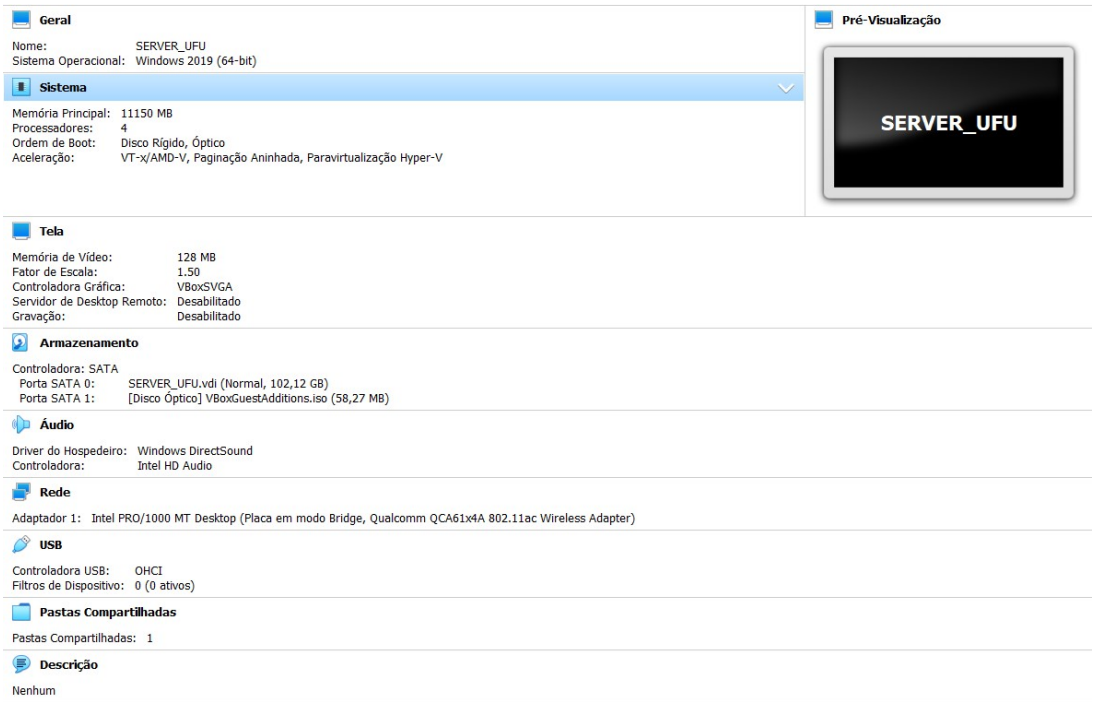

Figura 1 – Resumo de configurações da máquina virtual.

Fonte: Do autor

3.2.2 Configuração do IIS

O IIS é o servidor de aplicação utilizado para suportar a arquitetura atual do SCAD; de modo que para replicar a estrutura do ambiente real do SCAD foram criados

no IIS dois Sites, que nesse contexto se tratam de aplicações *web*. Isso foi necessário, pois a arquitetura do SCAD possui um *back-end* e um *front-end* que, apesar de funcionarem em conjunto, tem códigos fontes, tecnologias e propósitos diferentes. É importante destacar que cada um dos Sites no IIS opera de forma independente e permite que, por exemplo, se gerencie o uso de certificados SSL, o mapeamento de portas do protocolo HTTP, DNS, páginas de erro e arquivos internos.

A instalação de uma instância do IIS na máquina Windows Server foi realizada pelo *Server Manager*. O *Server Manager* é a aplicação responsável por gerenciar o servidor, a máquina Windows Server e os seus recursos. Essa aplicação provê um *Dashboard* que contém um menu intitulado *Add Roles and features*, e esse menu foi utilizado para instalar uma instância do IIS navegando por quatro telas de um assistente de instalação e selecionando o IIS na tela de *Server Roles*. Além disso, foi necessário habilitar os recursos do .NET Framework na tela seguinte, chamada *Features*. Estes últimos passos estão ilustrados nas Figuras 2 e 3.

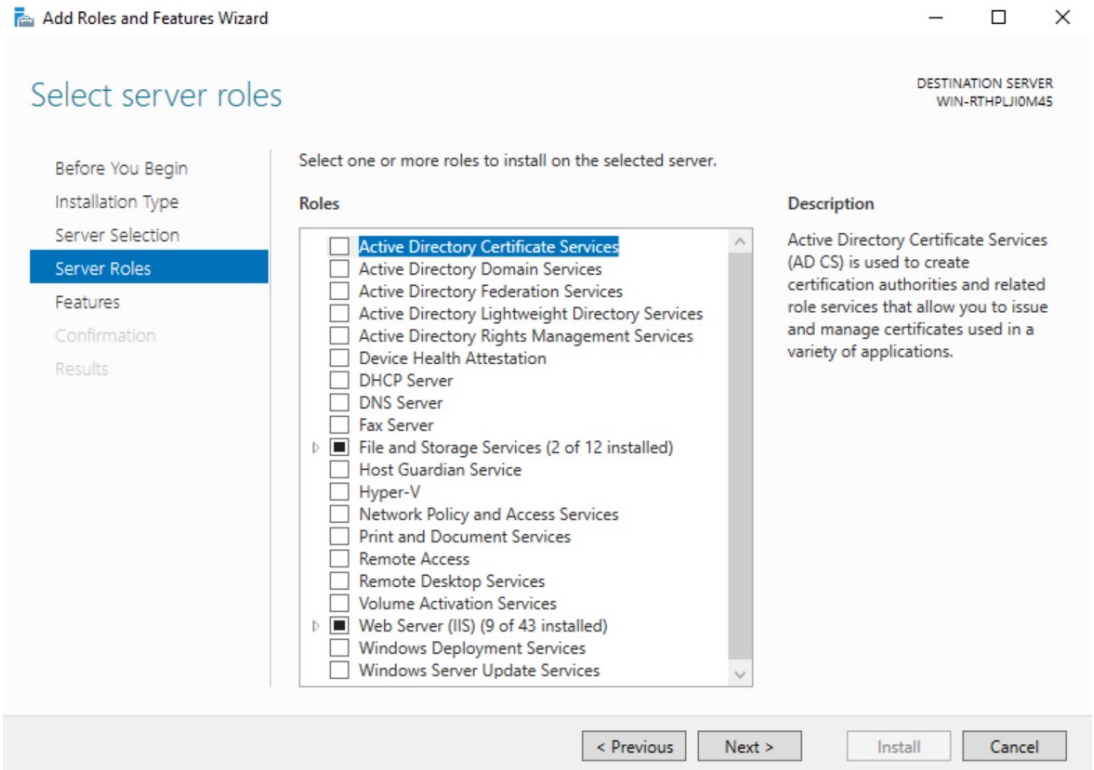

Figura 2 – Tela de *Server Roles*.

Fonte: Do autor

Como tratou-se de um ambiente de desenvolvimento, e não se faziam necessárias configurações diferentes das padrões para o servidor IIS, os dois Sites foram criados no seu contexto padrão, de modo que cada um deles foi mapeado para uma porta HTTP utilizando DNS padrão *(localhost)*. Os Sites foram nomeados scad e scadweb, o primeiro para abrigar o *back-end* e o segundo para o *front-end*.

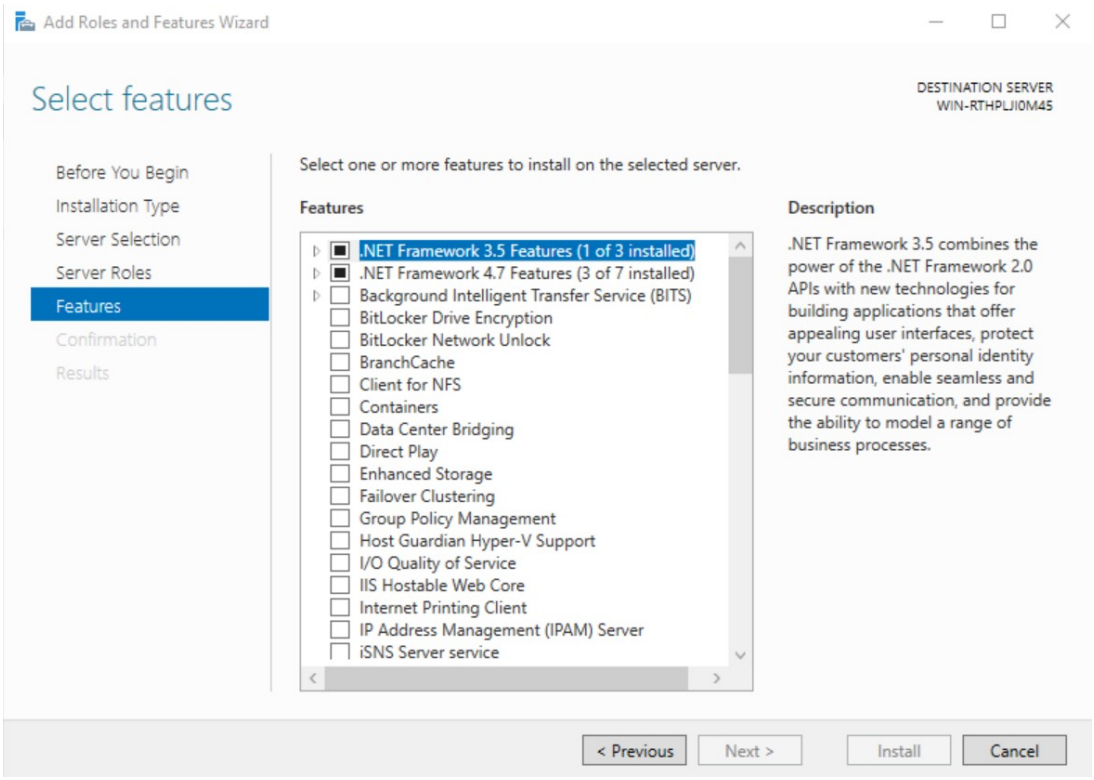

Figura 3 – Tela de *Features*.

Fonte: Do autor

A configuração do IIS foi finalizada com a implantação manual das aplicações em seus respectivos Sites, para que isso ocorresse os arquivos compilados, tanto do *back-end* quanto do *front-end*, foram inseridos nos diretórios que são criados pelo IIS para gerenciar os arquivos de uma aplicação.

3.3 Integração e entrega contínua com o uso Jenkins

O Jenkins é um servidor de automação que oferece os recursos e integrações necessárias para suportar um processo de integração contínua, como por exemplo, a integração com o Github e com o SonarQube. Trata-se de uma caixa de ferramentas poderosa para este tipo de demanda, pois tem uma grande quantidade de usuários e apesar de ser desenvolvido utilizando a plataforma JVM, oferece soluções para várias tecnologias como o C# e o Angular que são as empregadas no SCAD. As subseções abaixo tratam das nuances do trabalho realizado com essa que foi a principal ferramenta nesta criação do processo de integração e entrega contínua SCAD.

3.3.1 *Softwares* Estruturantes

O Jenkins é baseado na JVM, e portanto, precisa de uma para ser executado. Para isso foi necessário instalar no ambiente de desenvolvimento a JRE, que é o ambiente de

execução de programas que utilizam a JVM, a versão utilizada foi a JRE 11.

O Jenkins tem acesso e faz uso dos recursos presentes na máquina, e portanto, para realizar determinadas tarefas relativas ao processo que ele gerencia, alguns *softwares* foram instalados; como é o caso do Git, uma vez que os códigos se encontram em um repositório do GitHub, do Node pois o código do *front-end* foi construído com o Angular que é interpretado pelo Node, do MSBuild que é o *software* que a Microsoft distribui para realizar a compilação dos códigos escritos em C# utilizando o .NET Framework, e do WebDeploy que é a ferramenta do ecossistema da Microsoft responsável por realizar o *deploy* de aplicações no IIS.

3.3.2 Instalação do Jenkins

O *software* está disponível para *download* em seu site oficial (JENKINS, 2022) e o processo de instalação foi realizado no ambiente de desenvolvimento, em sua última versão estável. No assistente de instalação presente no executável distribuído, é possível configurar o diretório de destino dos arquivos do Jenkins, a porta HTTP através da qual o programa será acessado– comumente a porta utilizada para o Jenkins é a 8089– as credencias do usuário administrador, e a localização da JRE sob a qual este será executado. Uma vez cumpridas todas as etapas deste assistente de instalação, o Jenkins é instalado como um serviço no caso de uma máquina Windows, e fica disponível no endereço local, na porta escolhida. A tela inicial do Jenkins, após efetuado o *login*, pode ser observada na Figura 4.

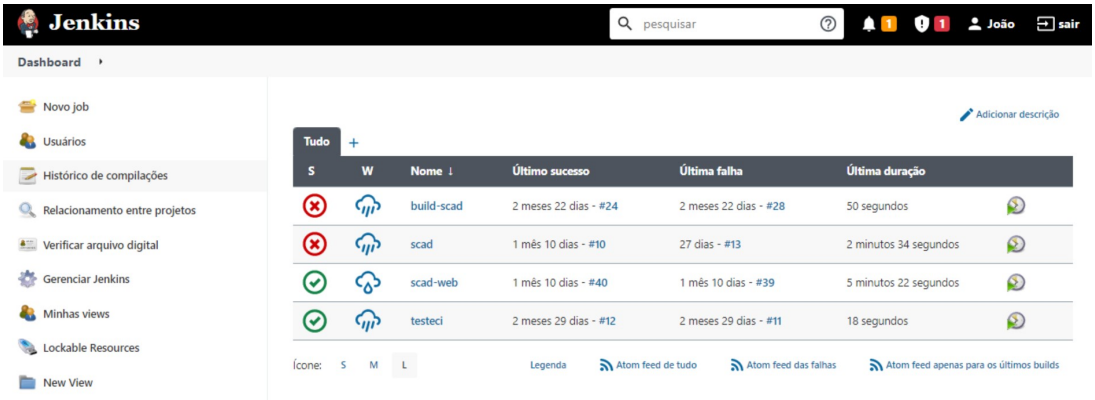

Figura 4 – Tela inicial do Jenkins.

Fonte: Do autor

3.3.3 Plugins

Para facilitar a integração do Jenkins com os recursos mencionados acima e expandir a capacidade do próprio *software*, este dispõe de uma vasta biblioteca de *plugins*,

um dos pontos positivos de um *software* de código aberto com uma comunidade grande, como é o caso do Jenkins.

Na instância do ambiente de desenvolvimento foram instalados os seguintes *Plugins*, disponíveis no Gerenciador de *Plugins do Jenkins*:

- .NET SDK Support, este *plugin* viabiliza a integração do Jenkins com as ferramentas do ecossistema Microsoft previamente instaladas.
- Generic Webhook Trigger, habilita a recepção de *webhooks* que são utilizados no contexto da esteira do SCAD para desencadear o processo quando há uma alteração no repositório do GitHub.
- Git, integra o Jenkins ao Git.
- GitHub, integra o Jenkins ao GitHub.
- MSBuild Plugin, integra o Jenkins ao MSBuild.
- NodeJs Plugin, possibilita a execução de comandos do Node no Jenkins.
- Pipeline: Groovy, oferece suporte à escrita de pipelines utilizando Groovy.
- SonarQube Scanner for Jenkins, integra o SonarQube ao Jenkins.

3.3.4 Pipelines

No Jenkins uma automação é chamada de *job*. O *job* pode assumir alguns arquétipos predefinidos, e para o caso do SCAD foi escolhido o chamado *Pipeline*, que nada mais é do que um *job* com uma sequência fixa de passos dependentes, ou seja, caso haja falha em algum passo, o próximo não é executado e o *job* como um todo falha. Como o SCAD é composto por duas aplicações, para cada uma delas foi criado um *pipeline*.

O que define os passos de um *job* é um *script* escrito em Groovy. Existem algumas funções pré definidas que podem ser utilizadas na escrita do *script* para o Jenkins, como é o caso da função *stage()*, que é usada para definir um passo do *pipeline*. Nela escreve-se em seu escopo o código que deverá ser executado em um determinado passo. Nas seções seguintes a escrita de cada um dos *pipelines* é detalhada.

Um recurso importante em um *job* do Jenkins é a demonstração visual da execução dos passos que são definidos. A partir dela é possível observar o tempo que cada um dos passos levou, é possível ser redirecionado, a partir de um clique, aos *logs* de cada um dos passos, observar um *feedback* por cor- vermelho para falhas e verde para sucesso- em qual passo houve falha, e o histórico de execuções com o tempo médio para o *job* em questão. Este elemento está ilustrado na Figura 5. Durante a fase de testes dos *pipelines* esse recurso ofereceu um bom suporte pois se trata de um *feedback* instantâneo da execução.

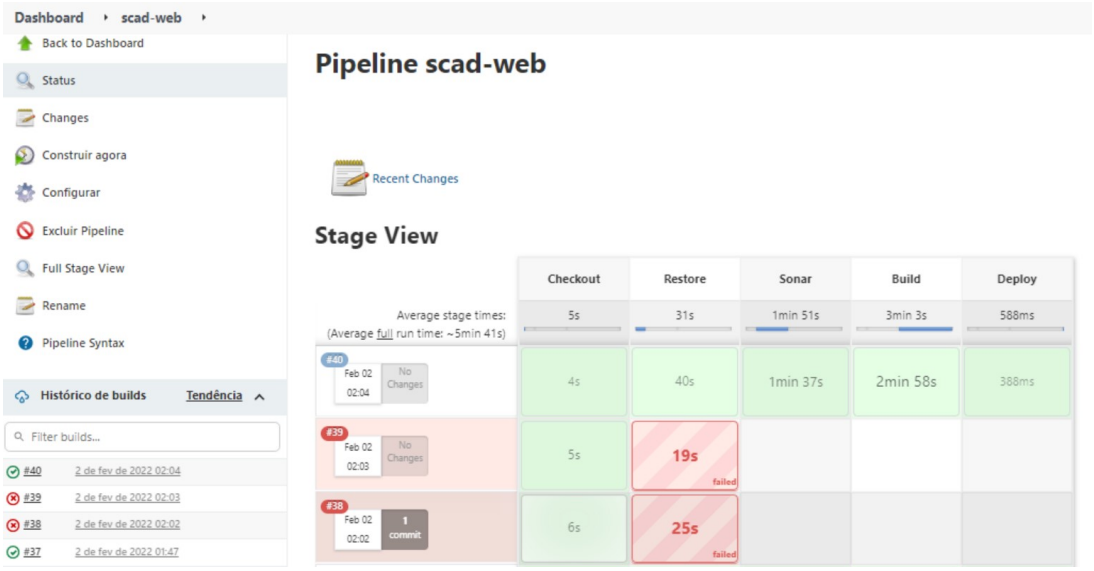

Figura 5 – Histórico de execução de um *job*.

Fonte: Do autor

3.3.4.1 *Back-End*

Para automatizar o processo do *back-end* o *pipeline* foi composto por quatro passos chamados de *Checkout*, *Sonar*, *Build* e *Deploy*.

No passo chamado *Checkout* foi escrito um *script*[1] utilizando a função *checkout()* que é predefinida pelo Jenkins e serve para realizar o *download* de um código fonte de um repositório Git remoto, através de parâmetros enviados para a função são especificados o endereço do repositório, a *branch* da qual se deseja baixar o código e as credenciais necessárias para acessá-lo.

O passo *Sonar* foi criado para executar o comando que gera o *report* e o envia para o SonarQube, que será discutido a seguir neste trabalho. Este passo se trata de um comando de *shell* e é necessário parametrizá-lo com as credenciais, o nome do projeto e o endereço do SonarQube.

O *Build* também utiliza um comando de *shell*, que executa a diretiva de construção do .NET Framework e publica em um diretório temporário.

Por fim, o *Deploy* é um comando *shell* que utiliza o recurso WebDeploy para enviar o pacote gerado para o *site* correspondente ao *back-end* no IIS. Neste passo foi utilizada uma função nativa do Jenkins chamada *withCredentials()* pois é necessário utilizar credenciais de usuário para executar o WebDeploy.

[1] https://github.com/JvitorA/scad-scripts/blob/main/scad-be.groovy

3.3.4.2 *Front-End*

Para automatizar o processo do *front-end*, o *pipeline*[2] foi criado utilizando cinco passos chamados de *Checkout*, *Restore*, *Sonar*, *Build* e *Deploy*.

O passo *Checkout* faz o mesmo que o descrito para o de mesmo nome no *back-end*. Com a diferença de que o endereço neste caso é o do repositório em que se encontra o código fonte do *front-end*.

No passo chamado *Restore* é executado via comando de *shell* a diretiva de instalação do Node, que é responsável por realizar o *download* das bibliotecas utilizadas no projeto, o Jenkins faz isso e guarda em um diretório temporário.

Após o *download* das bibliotecas o Node é capaz de executar um *plugin* chamado sonar-scanner, que é responsável por realizar a análise do código e enviar para o SonarQube, isto é feito no passo chamado *Sonar* do *pipeline* utilizando um comando de *shell*.

Para o passo *Build* foi escrito um comando de *shell* que usa a diretiva do Node para compilação; aqui são gerados os arquivos referentes à nova versão do código fonte.

O passo *Deploy* finaliza a esteira movendo os arquivos gerados para a pasta referente ao *site* do *front-end* no IIS com um comando *shell*. Como se tratam de arquivos interpretados e não verdadeiramente compilados, não houve necessidade de uma ferramenta de *deploy* como o WebDeploy neste caso, a substituição basta.

3.3.4.3 Execução dos *Pipelines*

Por padrão, o Jenkins oferece a opção de executar um *job* clicando em um botão disponível no *dashboard* do mesmo e também pela tela inicial do Jenkins na qual os *jobs* são listados. Uma outra maneira de iniciar a execução de um *job* é a utilização de gatilhos, que podem ser *webhooks* ou a finalização de um outro *job*, por exemplo.

Para o SCAD, em ambos os *jobs*, adotou-se o uso de *webhooks*, que são notificações enviadas de um sistema para outro com a intenção de informar o receptor de alguma mudança no emissor. Neste caso o Jenkins recebe uma notificação do GitHub de que houve um *push* no repositório remoto, logo, houve uma atualização do sistema e o processo de esteira é deve ser iniciado. Para que isso ocorresse nas configurações do *job* registrou-se a URL em que a notificação deverá ser recebida.

A fim de aumentar a segurança e evitar a exposição do Jenkins à internet foi utilizado um *proxy* chamado Webhook Relay, que é um *software* que roda uma instância local com conexão segura com a internet, pois precisa de um *token* para ser acessada. No GitHub configurou-se o envio do *webhook* para a instância do Webhook Relay, através de

[2] https://github.com/JvitorA/scad-scripts/blob/main/scad-fe.groovy

uma chamada devidamente autenticada, e ao receber a chamada o *software* redireciona a notificação para a URL registrada no Jenkins que só então inicia a esteira de execução. A política implementada no repositório é de só enviar essa notificação quando a alteração for realizada na branch *main*, pois para que essa seja atualizada é necessário um fluxo de aprovação do professor responsável pela administração do SCAD, o que mitiga as chances de uma execução indesejada da esteira no ambiente real. O processo descrito está ilustrado pela Figura 6, as linhas vermelhas indicam processos automatizados e as pretas manuais.

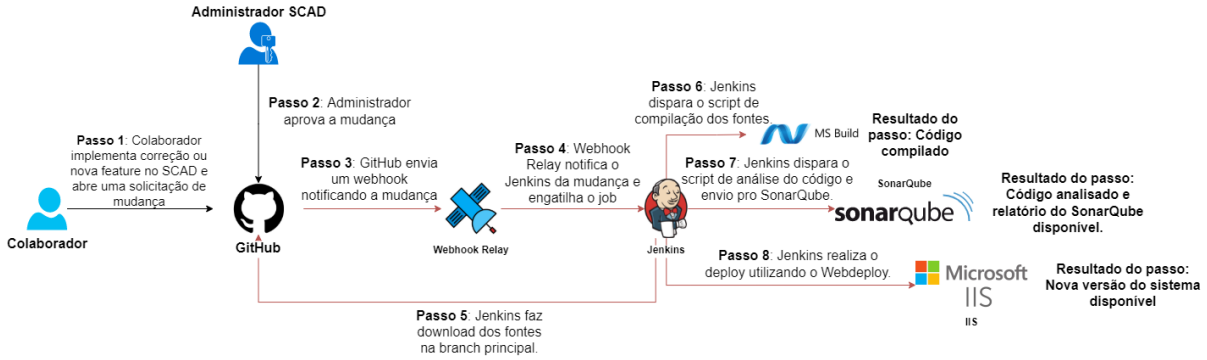

Figura 6 – Fluxograma de exemplo de execução de um *pipeline*.

Fonte: Do autor

3.4 Introdução do SonarQube

O SonarQube é um *software* que realiza a análise estática do código. Com o relatório gerado pela aplicação é possível observar pontos como a quantidade de bugs no código, pontos de vulnerabilidade e a cobertura de código por testes unitários. Essa ferramenta foi escolhida pois dá suporte nativo às duas linguagens base das aplicações que compõem o SCAD, além disso, existem *plugins* disponíveis para facilitar o processo de geração do relatório para cada uma delas e que foram utilizados na esteira do SCAD.

No contexto do SCAD, essa ferramenta foi inserida como um passo no processo de integração contínua em ambos os projetos nos seus respectivos *pipelines* do Jenkins. O objetivo dessa utilização foi identificar as possíveis vulnerabilidades em cada um dos projetos e os *bugs* reportados pela análise. Essa preparação possiblita que no longo prazo, por exemplo, crie-se uma regra de limites de bugs na análise para publicação de uma versão do SCAD, ou também que exija-se uma porcentagem mínima de cobertura de código por testes unitários para essa publicação.

3.4.1 Instalação e configuração do SonarQube

O SonarQube está disponível para *download* em uma versão gratuita chamada *Community*, que foi a utilizada no ambiente de desenvolvimento do SCAD. Assim como o

Jenkins o SonarQube é um *software* que precisa de uma JRE para ser executado, portanto, como ela já estava disponível na preparação do ambiente para o Jenkins não foi necessário instalá-la novamente no ambiente de desenvolvimento. Os passos da instalação descritos à seguir foram baseados na documentação do *software*[3].

A aplicação precisa de um banco de dados relacional para funcionar, pois utiliza este para armazenar os dados referentes aos projetos e relatório existentes. O *software* permite que se escolha entre o SQLServer, o Oracle e o PostgreSQL. Como o SCAD utiliza uma instância do PostgreSQL este foi o SGBD escolhido para o SonarQube no ambiente de desenvolvimento, pois foi necessário criar-se somente um *schema* para o SonarQube nessa instância. Foi necessário também editar o arquivo chamado *sonar.properties* que fica no diretório onde o SonarQube foi instalado, inserindo os parâmetros de conexão com o PostgreSQL.

Para executar a instância do SonarQube instalada, como se trata de uma máquina Windows, foi utilizado o *script* chamado *StartSonar.bat*, disponível no diretório de instalação do SonarQube. Após a execução do *script* o SonarQube pôde ser acessado através da porta HTTP 9000 no DNS local.

Após a instalação foram criados os projetos Scad e Scadweb para os códigos do *back-end* e do *front-end* respectivamente, utilizando o botão *New Project* presente na tela inicial do SonarQube, e que está representada na Figura 7. Posteriormente, foi adicionado aos *pipelines* do Jenkins o passo chamado *Sonar*, detalhado anteriormente neste texto, e a partir desse momento todas as execuções destes *pipelines* engatilham uma análise do código.

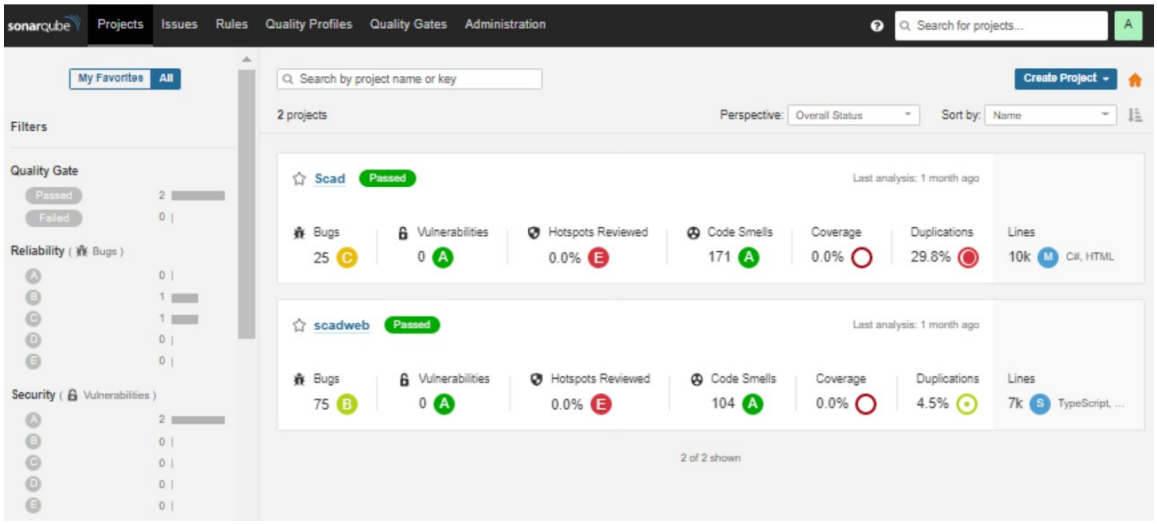

Figura 7 – *Dashboard* do SonarQube.

Fonte: Do autor

[3] https://docs.sonarqube.org/latest/setup/install-server/

3.4.2 Indicadores do SonarQube

A análise estática realizada pelo SonarQube aponta alguns indicadores à respeito do código, que são definidos pela documentação da ferramenta (SONARQUBE, 2022b) da seguinte maneira:

- ***Bugs***: Um ponto que representa um trecho de código que pode vir a quebrar a execução do sistema. Os *bugs* são subclassificados em *Info, Minor, Major* e *Blocker* de acordo com a sua severidade.
- ***Code Smells***: Quando há um possível problema de manutenção de um determinado trecho de código que não tem potencial de quebrar a execução do sistema mas que o torna confuso ou onera a sua performance, como por exemplo uma importação de biblioteca desnecessárias. Os *code smells* são subclassificados em *Info, Minor, Major, Critical* e *Blocker* de acordo com a sua severidade.
- ***Security Hotspots***: Quando a análise identifica um trecho de código que contém alguma possível vulnerabilidade de segurança que deve ser interpretada manualmente por quem o revisa, que vai decidir se é realmente uma ameaça ou não há necessidade de atuação. Os *security hotspots* são subclassificados em *Low, Medium* e *High* de acordo com a sua severidade.

Além de quantificar, a análise qualifica o apontamento, tipificando, apontando sugestões de solução para cada um deles, e estimando um tempo médio de correção como pode ser observado na Figura 8.

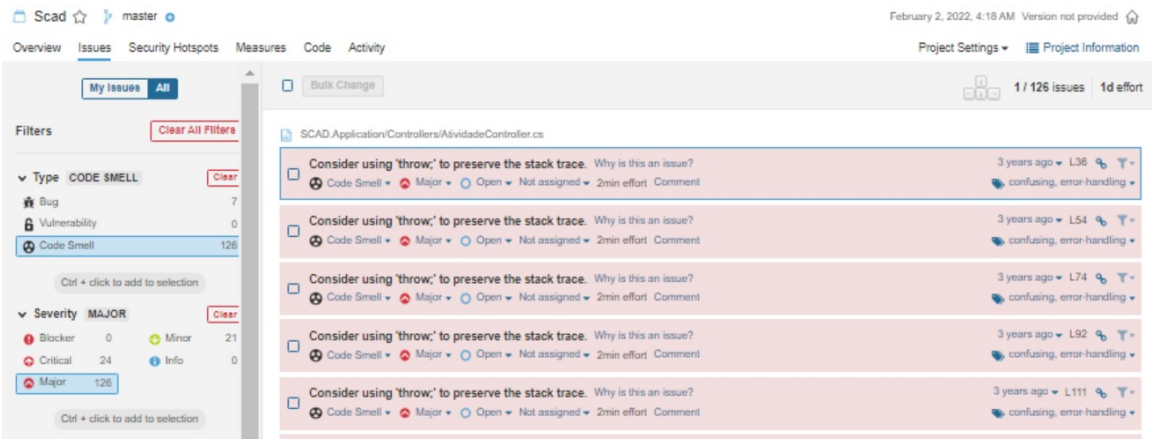

Figura 8 – Exemplo de qualificação de um apontamento.

Fonte: Do autor

3.4.3 Análise do SonarQube no código Back-End

A análise do código já existente para o *back-end* reportou 25 *bugs*, 2 *security hotspots* e 171 *code smells*, na Figura 9 é possível observar como o SonarQube apresenta estes resultados na página do projeto.

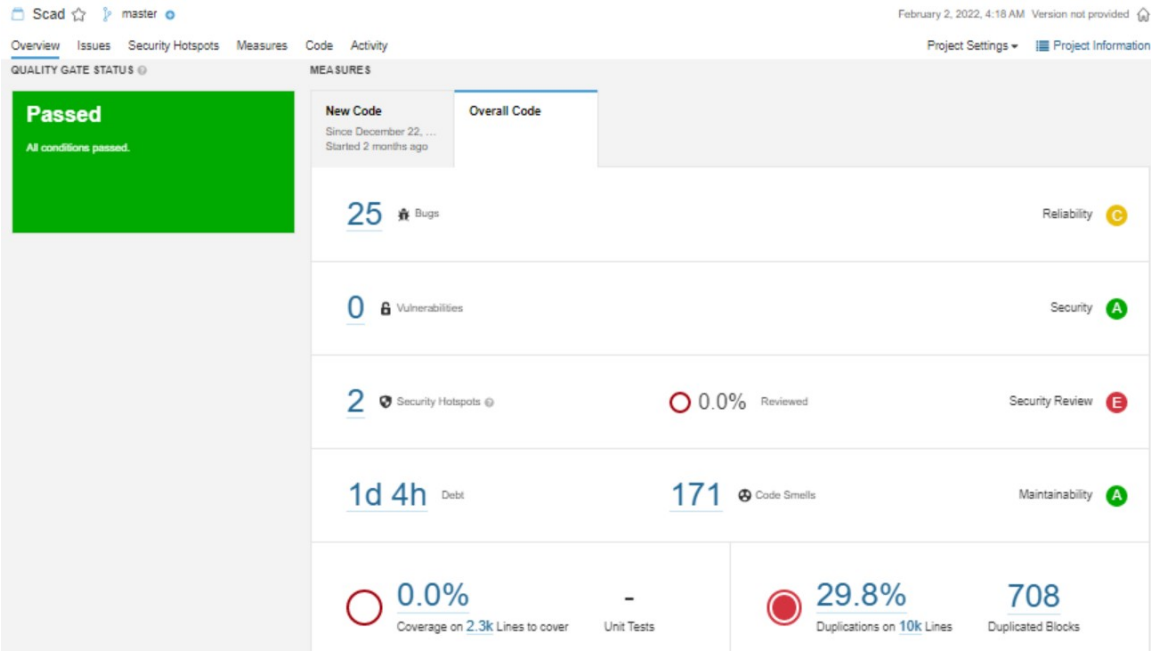

Figura 9 – *Dashboard* do projeto do *back-end*

Fonte: Do autor

No caso do código do *back-end* os dois *security hotspots* chamaram atenção, pois um se trata de uma geração de um número pseudoaleatório no processo de *hashing* da senha de um usuário no cadastro, classificado como *medium*. Já o segundo apontou uma política permissiva de CORS, que é um indicativo forte do uso do protocolo HTTP na sua forma insegura, esse ponto motivou a uma tarefa tratada à seguir neste trabalho.

Na Tabela 1 se discrimina os *bugs* e *code smells* presentes no código do *back-end* de acordo com sua severidade. Esses números, de acordo com os parâmetros da plataforma, não representam um grande problema mas podem ser objetos de um trabalho futuro com o fim de realizar a manutenção deste código.

Tabela 1 – Distribuição de *bugs* e *code smells* no *back-end*

	Minor	***Major***	***Critical***	***Blocker***
Bug	7	18	0	0
Code Smell	21	126	24	0

3.4.4 Análise do SonarQube no código Front-End

Ao analisar o código escrito para o *front-end* a ferramenta reportou 75 *bugs*, 3 *security hotspots* e 104 *code smells*, na Figura 10 pode-se observar esse relatório.

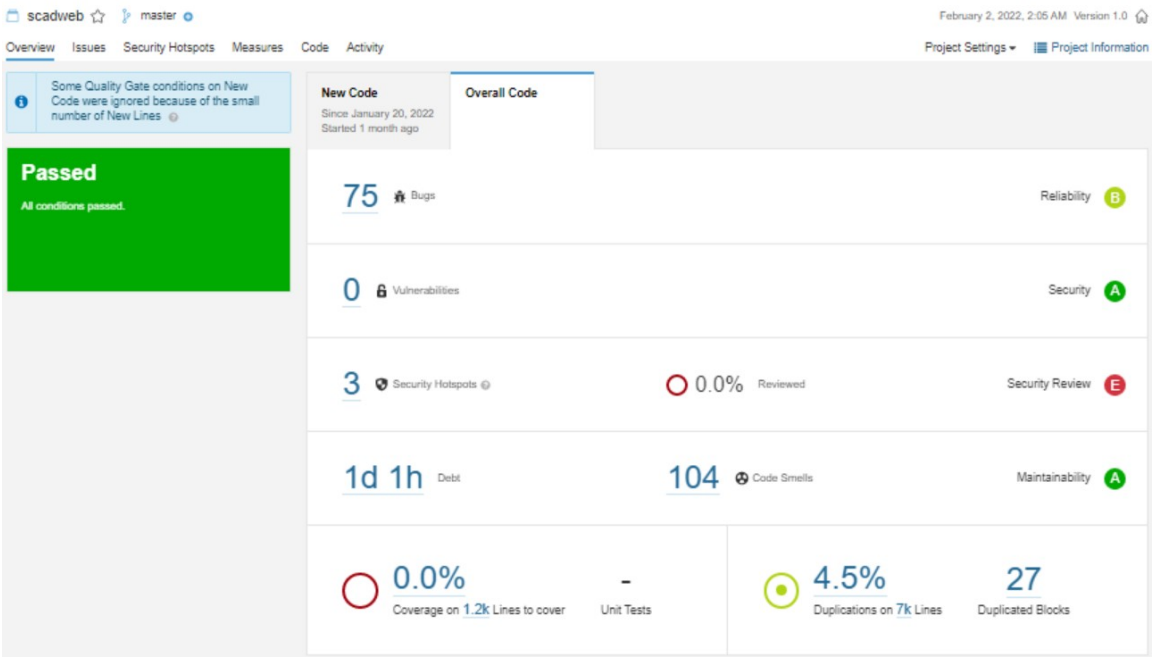

Figura 10 – *Dashboard* do projeto do *front-end*

Fonte: Do autor

No caso do código do *front-end* destacaram-se 3 *security hotspots*, dois deles se tratam do uso de *regex* que pode tornar o código vulnerável, no caso do *front-end* à um ataque de *Denial of Service*. O último apontamento é análogo ao presente no relatório do *back-end*, que é uma política permissiva de CORS, habilitando qualquer origem, reforçando novamente a necessidade da implantação do HTTPS e a definição correta das origens aceitas.

Na Tabela 2 se discrimina os *bugs* e *code smells* presentes no código do *front-end* de acordo com sua severidade. Neste caso os números também não representam algo alarmante, na avaliação da plataforma, mas que também abrem a possibilidade de um trabalho de manutenção no código da aplicação.

Tabela 2 – Distribuição de *bugs* e *code smells* no *front-end*

	Minor	***Major***	***Critical***	***Blocker***
Bug	75	0	0	0
Code Smell	27	45	32	0

3.5 Melhorias de Segurança

Uma outra frente de trabalho foi implantar melhorias na segurança do SCAD. Uma delas, a restrição da política de CORS, foi uma consequência direta da implantação da esteira de integração contínua, visto que a vulnerabilidade foi apontada durante a análise do SonarQube. Já a necessidade da implantação do HTTPS foi identificada através da análise manual em busca de vulnerabilidades conhecidas.

3.5.1 Restrição da política de CORS

Os navegadores modernos implementam a chamada *same-origin policy*, ou política de mesma origem, o que quer dizer que numa página da web só se podem enviar requisições para própria origem, por exemplo um site cujo domínio é *siteexemplo.com* só pode, dentro dessa política, enviar requisições para endereços que estejam nesse mesmo domínio, como por exemplo requisitar uma imagem em *siteexemplo.com/bichinhofeliz.jpg*. Essa política surgiu para evitar ataques que carregam os cookies para um outro site através de um *link* malicioso e disparam requisições para a origem real utilizando uma sessão logada capturada desses *cookies*. Essa política considera como origem diferente, ou origem cruzada, requisições que venham de uma porta diferente, de um subdomínio diferente ou utilizem um protocolo diferente. A Figura 11 exemplifica esse ponto. O desafio que essa política trás consigo é o de lidar com casos como o do SCAD, que são bem comuns, em que se tem uma aplicação *front-end* em um domínio que consome uma API *back-end* de um outro domínio. Para isso existe o que chamamos de *cross origin resource sharing*, ou compartilhamento de recursos de origem cruzada, ou simplesmente CORS, que é uma maneira segura de estabelecer uma comunicação entre origens diferentes. Esse mecanismo se utiliza de cabeçalhos nas requisições para prover informações ao navegador, como por exemplo o *Access-Control-Allow-Origin* que deve estar presente na resposta emitida pelo servidor que recebe a requisição e conter as origens que podem enviar essa determinada requisição e, caso a origem da requisição esteja presente nessa lista, o navegador permite que ela seja completada.

No processo de análise do código do SCAD com o uso do SonarQube foi identificado que as políticas de CORS estavam permissivas, considerado na classificação da ferramenta um *security hotspot*. Uma política permissiva, no que diz respeito ao CORS, quer dizer que é passado no cabeçalho *Access-Control-Allow-Origin* a seguinte expressão '*', que faz com que o servidor aceite requisições de qualquer origem, e receber requisições de qualquer origem é o mesmo que ignorar a existência e a segurança fornecida pela política de mesma origem. Dado que as aplicações estão hospedadas em um servidor IIS, foi possível sobrepor as configurações de CORS no código das aplicações, isso foi feito instalando o módulo de CORS no IIS e adicionando ao arquivo *web.config* do projeto do *back-end* a *tag* <*cors*> e permitindo somente a URL do *front-end* como origem.

Figura 11 – Exemplificação da política de mesma origem

Fonte: Do autor

3.5.2 Implantação do HTTPS

Uma vulnerabilidade bastante conhecida dos sistemas web é a opção pelo uso do protocolo HTTP na sua forma insegura, sem a camada de criptografia implementada pelo HTTPS. De uma maneira simples, essa exposição possibilita que as trocas de mensagens utilizando o protocolo sejam intereceptadas e tenham os seus conteúdo expostos para quem o fez, já que nessa versão do protocolo as mensagens são trocadas livremente. Uma vez que o HTTPS introduz uma camada de criptografia nas mensagens trocadas, o interceptador pode até ter acesso à mensagem, contudo, vai encontrá-la cifrada e sem utilidade alguma para uma possível exploração, portanto o uso do HTTPS tem sido largamente incentivado até mesmo para sistemas que não trocam mensagens sensíveis.

Ao identificar que tanto o *front-end* quanto a API foram expostos usando o protocolo HTTP, foram cumpridos alguns passos para implantar no SCAD o protocolo HTTPS. A primeira ação foi gerar um certificado, no caso do ambiente de testes foi utilizada a ferramenta *mkcert*, que cria localmente uma entidade certificadora, de forma que os navegadores não considerem que este se trata de um certificado auto-assinado, a ferramenta

tem seu código aberto e está disponível no GitHub[4]. Para a implantação no servidor real, que se encontra exposto à Internet, será necessário trocar esse por um certificado assinado por uma entidade certificadora real. Após a geração do certificado foram realizados ajustes nas configurações no IIS, o primeiro foi instalar o certificado através do menu *server certificates*, depois foram modificadas as portas de exposição dos sites para 443 e 444, pois se tratam das portas utilizadas pelo protocolo. Por último, no código do *front-end* foi alterada a variável de ambiente referente a URL padrão da API do *back-end*, para utilizar o prefixo *https://* em detrimento do *http://*, isso foi feito editando o arquivo *environment.prod.ts*.

[4] https://github.com/FiloSottile/mkcert

4 Conclusão

Ao analisar o trabalho realizado concluiu-se que o objetivo geral e os objetivos específicos foram atingidos. Nos parágrafos à seguir são listados os benefícios trazidos para as diferentes personas presentes no ecossistema do SCAD após a realização deste trabalho.

O primeiro benefício foi que o processo de integração contínua otimiza muitas questões atreladas ao anterior, feito de forma manual, pois economiza o tempo do responsável pelo sistema que tinha de acessar o servidor e realizá-lo, mitiga a possibilidade de uma falha humana no manejo dos arquivos e na execução dos comandos necessários para compilá-los. A partir dessa implantação até mesmo o processo de desenvolvimento de novas funcionalidades para o sistema foi desburocratizado, já que a entrega de uma funcionalidade passa a depender somente da aprovação do responsável dentro dos requisitos que ele julgar necessário, também se torna mais simples o processo de reversão de uma mudança indesejada, haja vista que esse processo tinha o mesmo custo, em termos de trabalho, que o de entrega e agora é automático.

A implantação de uma ferramenta de análise estática de código, o SonarQube, ao decorrer da esteira de integração tornou possível identificar na base de código do SCAD onde se encontram pontos sensíveis e de melhoria do mesmo. Ao identificar esses pontos este trabalho viabilizou trabalhos futuros que tenham como objetivo efetuar as devidas correções nos pontos encontrados.

Já as melhorias de segurança implementadas tiveram um impacto direto no usuário final, no ponto em que os seus dados passaram a estar mais protegidos pois está sendo trafegados via HTTPS. Além disso, o administrador do sistema passou a não ter que lidar com algumas ameaças de segurança como por exemplo os ataques *man in the middle* que eram facilitados pela política permissiva de CORS que existia previamente.

Por último, utilizando a base ferramental implementada por este trabalho é possível vislumbrar trabalhos futuros como por exemplo a cobertura do código com testes unitários por hora apontada como 0% pelo SonarQube, medida que vai aumentar a confiabilidade do código e possibilitar o desenvolvimento de novas funcionalidades sem que se preocupe com o impacto no que já existe, a correção dos *bugs e code smells* apontados pelo SonarQube a fim de eliminar o débito técnico presente no código e também o reforço na segurança da aplicação eliminando os *security hotspots*.

Referências

ANGULAR. 2022. *Documentação do Angular 2*. Disponível em: <https://v2.angular.io/docs/ts/latest/>. Acesso em: 08 fevereiro 2022. Citado na página 13.

FILOSOTTILE. 2022. *Repositório do Mkcert*. Disponível em: <https://github.com/FiloSottile/mkcert>. Acesso em: 20 março 2022. Citado na página 13.

FORSGREN, N.; SMITH, D.; HUMBLE, J.; FRAZELLE, J. 2019 accelerate state of devops report. 2019. Citado na página 11.

GROOVY. 2022. *Documentação da Linguagem Groovy*. Disponível em: <http://groovy-lang.org/documentation.html>. Acesso em: 08 fevereiro 2022. Citado na página 13.

JENKINS. 2022. *Documentação do Jenkins*. Disponível em: <https://www.jenkins.io/doc/>. Acesso em: 22 fevereiro 2022. Citado 2 vezes nas páginas 13 e 18.

MICROSOFT. 2022. *Documentação do C#*. Disponível em: <https://docs.microsoft.com/pt-br/dotnet/csharp/>. Acesso em: 08 fevereiro 2022. Citado na página 12.

_____. 2022. *Documentação do .NET Core*. Disponível em: <https://docs.microsoft.com/pt-br/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-3.1>. Acesso em: 08 fevereiro 2022. Citado na página 13.

_____. 2022. *Documentação do IIS*. Disponível em: <https://docs.microsoft.com/en-us/iis/get-started/getting-started-with-iis/getting-started-with-the-iis-manager-in-iis-7-and-iis-8>. Acesso em: 08 fevereiro 2022. Citado na página 13.

_____. 2022. *Repositório do WebDeploy*. Disponível em: <https://www.iis.net/downloads/microsoft/web-deploy>. Acesso em: 20 março 2022. Citado na página 13.

_____. 2022. *Download da ISO do Windows Server*. Disponível em: <https://www.microsoft.com/pt-br/evalcenter/evaluate-windows-server-2019>. Acesso em: 08 fevereiro 2022. Citado na página 15.

ORACLE. 2022. *Documentação do VirtualBox*. Disponível em: <https://www.virtualbox.org/wiki/Documentation>. Acesso em: 08 fevereiro 2022. Citado na página 13.

POSTGRESQL. 2022. *Documentação do PostgreSQL*. Disponível em: <https://www.postgresql.org/docs/current/intro-whatis.html>. Acesso em: 08 fevereiro 2022. Citado na página 13.

REIS, M. P. 2021. *Sistema de informação para gerenciamento de progressão e promoção funcional*. Disponível em: <https://repositorio.ufu.br/handle/123456789/33841>. Acesso em: 08 fevereiro 2022. Citado 2 vezes nas páginas 10 e 11.

SILVA, W. S. 2019. *Desenvolvimento do sistema de cadastro de atividades docente*. Disponível em: <https://repositorio.ufu.br/handle/123456789/28962>. Acesso em: 08 fevereiro 2022. Citado na página 10.

SONARQUBE. 2022. *Documentação do SonarQube*. Disponível em: <https://docs.sonarqube.org/latest/>. Acesso em: 22 fevereiro 2022. Citado na página 13.

_____. 2022. *Documentação do SonarQube - Conceitos*. Disponível em: <https://docs.sonarqube.org/latest/user-guide/concepts/>. Acesso em: 22 fevereiro 2022. Citado na página 24.

STALLINGS, W. **Cryptography and network security principles and practices 4th edition**. [S.l.]: Pearson Education, Inc, 2006. Citado na página 11.

WEBHOOKRELAY. 2022. *Documentação do Webhook Relay*. Disponível em: <https://docs.webhookrelay.com/>. Acesso em: 22 fevereiro 2022. Citado na página 13.

Apêndices

APÊNDICE A – *Troubleshooting* na configuração do ambiente

À seguir estão listadas algumas situações que podem ocorrer durante a instalação, configuração do ambiente e execução da esteira de integração contínua seguidas de um direcionamento para contorná-las.

A.1 Credenciais inválidas durante a instalação do Jenkins

Em um dos passos de instalação é necessário introduzir um usuário para que o Jenkins seja executado como um serviço, e caso este usuário não tenha permissão de *logon* como um serviço será mostrado o erro indicado na Figura 12. A solução é atribuir à esse usuário esses privilégios, as instruções específicas estão disponíveis no manual de instalação do Jenkins pra Windows[1].

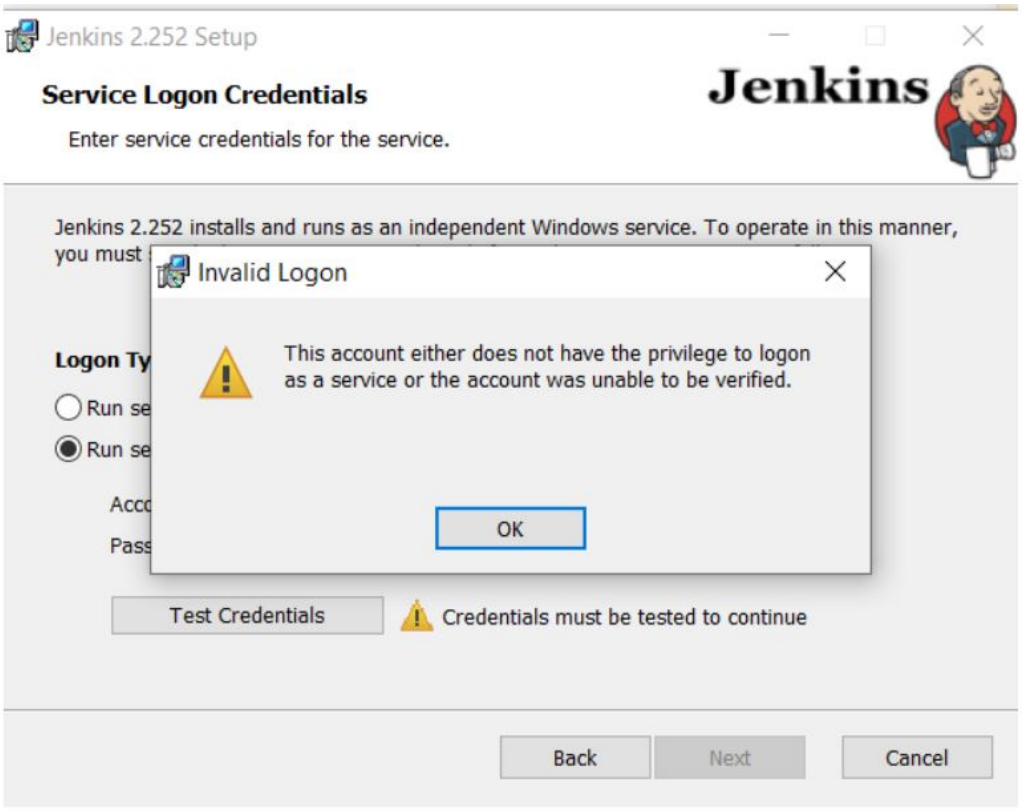

Figura 12 – Captura da tela do erro de credenciais inválidas

Fonte: Do autor

[1] https://www.jenkins.io/doc/book/installing/windows/

A.2 Execução dos *scripts*

A.2.1 *Back-End*

A.2.1.1 *Checkout*

É necessário criar um par de credenciais dentro do Jenkins[2] para alimentar a variável *credentialsID* que é usada nesse passo da execução, uma vez que ela é criada por ambiente. Após criada a credencial deverá ser atribuída à variável o respectivo identificador, que será um *UUID*.

A.2.1.2 *Sonar*

Este cenário pode apresentar erro caso o *plugin* não esteja instalado e devidademente configurado. As instruções de instalação do *plugin* estão disponíveis em sua documentação[3]. É importante o arquivo *SonarQube.Analysis.xml* esteja devidamente preenchido conforme o ambiente em que o sistema esteja sendo instalado.

Outro ponto necessário para a execução correta deste passo é a criação de um *token* de acesso dentro do SonarQube e a alteração da variável *d:sonar.login* para o valor do *token* criado.

A.2.2 *Front-End*

A.2.2.1 *Checkout*

Para evitar erros de permissionamento também é necessário alimentar a variável *credentialsId*, como foi feito no passo descrito na Seção A.2.1.1.

A.3 Jenkins não reconhece o Node

Ao instalar o Node na máquina pode ser que o Jenkins não o reconheça em um primeiro momento, para solucionar essa questão o Jenkins pode ser reiniciado através da URL *http://localhost:portaConfigurada/safeRestart*.

A.4 Habilitar o Serviço de Gerenciamento

O serviço de gerenciamento é um requisito para o funcionamento do WebDeploy, para habilitá-lo navegue para o Assisente de Adição de Funções e Recursos do Windows e Habilite o Serviço de Gerenciamento dentro da aba Ferramentas de Gerenciamento, como indicado na Figura 13.

[2] https://gago.io/blog/jenkins-pipeline-secret-files/
[3] https://docs.sonarqube.org/latest/analysis/scan/sonarscanner-for-msbuild/

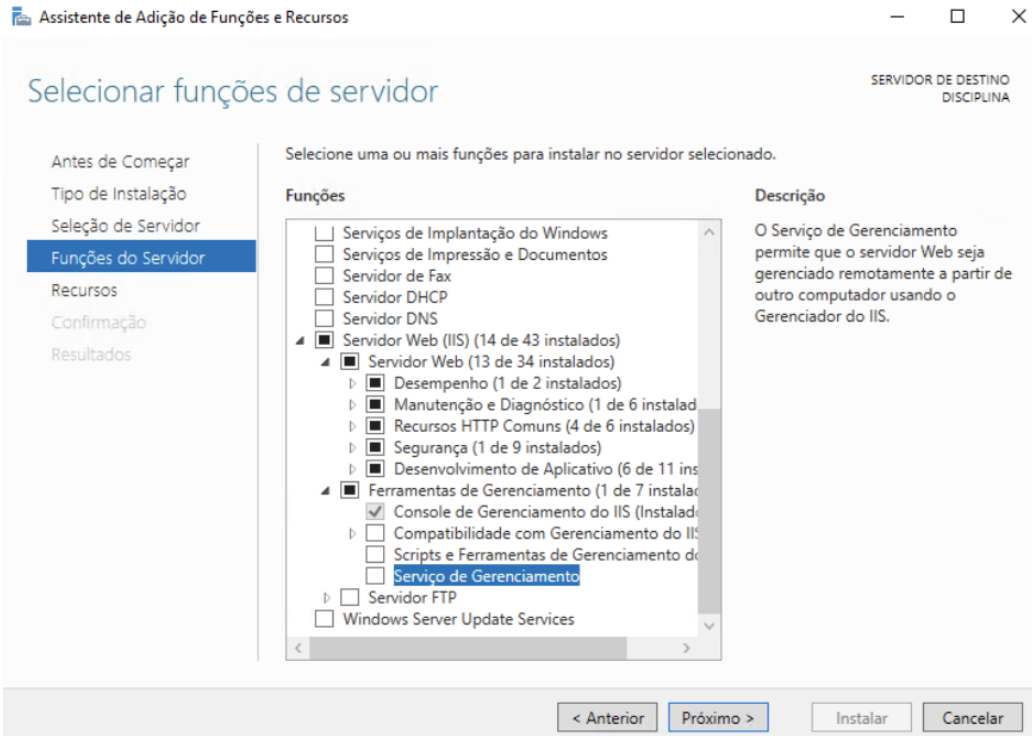

Figura 13 – Captura da tela do Assisente de Adição de Funções e Recursos do Windows

Fonte: Do autor