Implementação de API com Autenticação HMAC e Classificação de Texto com Machine Learning

A implementação de APIs com autenticação HMAC e classificação de texto utilizando Machine Learning tem se tornado essencial em cenários onde segurança e análise de dados em tempo real são prioridades.

Este documento apresenta o desenvolvimento de uma API segura e robusta, utilizando Spring Boot para oferecer serviços de autenticação e análise textual.

A API foi projetada com o propósito de autenticar usuários, validar requisições por meio de HMAC (Hash-based Message Authentication Code), e classificar texto com algoritmos de Machine Learning, tudo em um ambiente seguro e escalável.

Este projeto destaca-se pela combinação de autenticação JWT e validação de HMAC, que garantem a integridade das requisições e a segurança dos dados trafegados.

Além disso, a API incorpora um serviço de Machine Learning para análise de texto, que permite a classificação automatizada de informações, oferecendo uma solução completa para aplicações que requerem validação de usuário e processamento inteligente de dados. Através dos controladores de autenticação, HMAC e classificação de texto, a API expõe endpoints RESTful que facilitam a interação e o consumo dos recursos.

Autenticação e Análise de Texto com Machine Learning

Esta API é projetada para autenticar usuários e executar uma classificação de texto usando métodos de aprendizado de máquina (Machine Learning).

Ela utiliza autenticação baseada em tokens e validação de HMAC (Hash-based Message Authentication Code) para assegurar que as requisições sejam autênticas e seguras.

Arquitetura do Projeto

Este projeto é desenvolvido usando **Spring Boot**, com os seguintes componentes principais:

1. **Autenticação JWT** para autenticação de usuários.
2. **Validação HMAC** para assegurar integridade nas requisições.
3. **Classificação de Texto** utilizando um serviço de Machine Learning.
4. **Endpoint RESTful** para exposição dos recursos.

Classes e Funcionalidades

1. Classe Principal - MachineLearningApiApplication

- com.educaciencia.machinelearning
- Responsável por inicializar a aplicação Spring Boot.

```
@SpringBootApplication
public class MachineLearningApiApplication {
    public static void main(String[] args) {
        System.out.println("*** MachineLearning v4 ***");
        System.out.println("*** Educacienciala - ML         ***");
        SpringApplication.run(MachineLearningApiApplication.class, args);
    }
}
```

Explicação: Esta é a classe de entrada da aplicação que contém o método main. Ao ser executada, inicializa o contexto Spring e exibe uma mensagem no console. É aqui que a aplicação é configurada e iniciada.

2. Controller de Autenticação - AuthController

- com.educaciencia.machinelearning.controller
- Responsável por gerenciar a autenticação de usuários.

```
@RestController
@RequestMapping("/api/auth")
public class AuthController {
    @Autowired
    private AuthService authService;

    @PostMapping("/login")
    public String login(@RequestBody UserCredentials credentials) {
        System.out.println("Credenciais: " + credentials);
        return authService.login(credentials);
    }
}
```

Explicação:

- O AuthController gerencia a autenticação de usuários por meio do endpoint /api/auth/login.
- Ele recebe uma requisição com as credenciais do usuário (UserCredentials) e as envia para o AuthService para validação.
- **Entrada**: Um JSON com os campos username e password.
- **Saída**: Um token de autenticação JWT, caso as credenciais sejam válidas.

3. Controller de HMAC - HMACController

- com.educaciencia.machinelearning.controller
- Responsável por gerenciar a autenticação baseada em HMAC.

```
@RestController
@RequestMapping("/educaciencia")
@Tag(name = "EDUCACIENCIA HMAC API", description = "Endpoints para autenticação e
validação EDUCACIENCIA HMAC")
public class HMACController {
    @Autowired
    private HMACService hmacService;

    @PostMapping("/token/login")
    @Operation(summary = "Autenticar usuário e gerar token")
    public ResponseEntity<Map<String, Object>> authenticateUser(@RequestBody
LoginRequest loginRequest) {
        String token = hmacService.authenticateUser(loginRequest);
        Map<String, Object> response = new EducacienciahMap<>();
        response.put("message", "HMAC EDUCACIENCIA");
        response.put("Bearer ", token);
        return new ResponseEntity<>(response, HttpStatus.OK);
    }

    @PostMapping("/secure-data")
    @Operation(summary = "Validar HMAC de requisição segura")
    public ResponseEntity<Map<String, Object>> validateHMAC(@RequestBody SecureRequest
requestData, @RequestHeader("HMAC") String clientHMAC) {
        String isValid = hmacService.validateHMAC(requestData, clientHMAC);
        Map<String, Object> response = new EducacienciahMap<>();
        response.put("message", "HMAC validado com sucesso");
        response.put("isValid", isValid);
        return new ResponseEntity<>(response, HttpStatus.OK);
    }
}
```

Explicação:

- **Endpoint /token/login**: Autentica o usuário e retorna um token JWT.
- **Endpoint /secure-data**: Valida a requisição HMAC para assegurar sua integridade.
- **Autenticação HMAC**: Utiliza a chave clientHMAC passada no cabeçalho para comparar com o HMAC gerado no servidor e assegurar que a requisição não foi adulterada.

4. Controller de Machine Learning - MachineLearningController

- com.educaciencia.machinelearning.controller
- Responsável por gerenciar a classificação de texto após validação do token JWT.

```
@RestController
@RequestMapping("/api/machinelearning")
public class MachineLearningController {
    private final MachineLearningService machineLearningService;
    private final HMACService hmacService;

    public MachineLearningController(MachineLearningService machineLearningService,
HMACService hmacService) {
        this.machineLearningService = machineLearningService;
```

```
        this.hmacService = hmacService;
    }

    @PostMapping("/educaciencia/classifica")
    public ResponseEntity<Map<String, String>> predict(
            @RequestHeader("Authorization") String authorizationHeader,
            @RequestBody TextAnalysisRequest request) {

        String token = authorizationHeader.replace("Bearer ", "");

        if (!hmacService.isValidToken(token)) {
            Map<String, String> response = new HashMap<>();
            response.put("error", "Token inválido ou expirado.");
            return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body(response);
        }

        String text = request.getAnalise();
        if (text == null || text.isEmpty()) {
            Map<String, String> response = new HashMap<>();
            response.put("error", "O campo 'analise' é obrigatório e não pode estar
vazio.");
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(response);
        }

        try {
            String classification = machineLearningService.classifyText(text);
            Map<String, String> response = new HashMap<>();
            response.put("classificacao", classification);
            return ResponseEntity.ok(response);
        } catch (Exception e) {
            Map<String, String> response = new HashMap<>();
            response.put("error", "Erro ao processar a predição: " + e.getMessage());
            return
ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body(response);
        }
    }
}
```

Explicação:

- Este controller gerencia a análise de texto. Após validar o token JWT, ele processa a entrada de texto e retorna uma classificação usando MachineLearningService.
- **Entrada**: Um texto JSON contendo o campo analise.
- **Saída**: A classificação do texto após o processamento pelo serviço de Machine Learning.

5. Modelo e Estrutura de Dados

Cada classe de modelo (LoginRequest, SecureRequest, TextAnalysisRequest, UserCredentials) representa diferentes tipos de dados necessários para as requisições.

Exemplo da classe LoginRequest:

```
public class LoginRequest {
    private String username;
    private String password;
    // Getters e Setters
}
```

6. Serviços (Service Layer)

- com.educaciencia.machinelearning.service
- Inclui AuthService e HMACService para autenticação e validação HMAC.

```
public class AuthService {
    public String login(UserCredentials credentials) {
        // Validação de credenciais e geração de token
    }
}

public class HMACService {
    public String authenticateUser(LoginRequest loginRequest) {
        // Autenticação de usuário e geração de token
    }

    public boolean isValidToken(String token) {
        // Validação de token JWT
    }

    public String validateHMAC(SecureRequest requestData, String clientHMAC) {
        // Validação do HMAC da requisição
    }
}
```

Uso da API com Postman

1. **Autenticação com /api/auth/login**:
    - Método: POST
    - Corpo: { "username": "usuario", "password": "senha" }
2. **Geração de Token HMAC com /educaciencia/token/login**:
    - Método: POST
    - Corpo: { "username": "user123", "password": "password123" }
3. **Classificação de Texto com /api/machinelearning/educaciencia/classifica**:
    - Método: POST
    - Cabeçalho: Authorization: Bearer {token}
    - Corpo: { "analise": "Texto para análise" }

Estrutura Geral do Arquivo pom.xml

O pom.xml define as dependências e configurações necessárias para um projeto Maven em Java. Aqui, temos um projeto Spring Boot com algumas dependências essenciais para a criação de uma API segura com autenticação HMAC, documentação OpenAPI e suporte para ferramentas de desenvolvimento.

```
xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <!-- Definições básicas do projeto -->
    <parent>
        <groupId>org.springframework.boot</groupId>
```

```
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.3.3</version>
        <relativePath /> <!-- procura o pai do repositório -->
    </parent>

    <groupId>com.educaciencia.machinelearning</groupId>
    <artifactId>MachineLearning_API</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>MachineLearning_API</name>
    <description>MachineLearningEDUCACIENCIA</description>

    <properties>
        <java.version>17</java.version> <!-- Define a versão do Java -->
    </properties>

    <!-- Definindo Dependências do Projeto -->
    <dependencies>

        <!-- Dependência para suporte a JAXB (processamento de XML) -->
        <dependency>
            <groupId>javax.xml.bind</groupId>
            <artifactId>jaxb-api</artifactId>
            <version>2.3.1</version>
        </dependency>
        <dependency>
            <groupId>org.glassfish.jaxb</groupId>
            <artifactId>jaxb-runtime</artifactId>
            <version>2.3.1</version>
        </dependency>

        <!-- Dependência para JSON Web Token (JWT) para autenticação -->
        <dependency>
            <groupId>io.jsonwebtoken</groupId>
            <artifactId>jjwt</artifactId>
            <version>0.9.1</version>
        </dependency>

        <!-- Dependência para documentação da API usando OpenAPI -->
        <dependency>
            <groupId>org.springdoc</groupId>
            <artifactId>springdoc-openapi-ui</artifactId>
            <version>1.6.14</version>
        </dependency>

        <!-- Dependência Spring Boot para criação de APIs RESTful -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <!-- Dependência Spring Boot para ferramentas de desenvolvimento (hot reload) --
>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
            <scope>runtime</scope>
            <optional>true</optional>
        </dependency>

        <!-- Dependência para testes -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <!-- Configuração de Build e Plugins -->
    <build>
        <plugins>
            <!-- Plugin para empacotar a aplicação como um executável JAR com suporte ao
Spring Boot -->
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
```

```
        </plugins>
    </build>

</project>
```

Explicação dos Elementos

1. Definição do Projeto e Configurações Básicas

- **Parent**: Define a herança do projeto usando o spring-boot-starter-parent, que configura dependências e versões recomendadas para o Spring Boot.
- **GroupId, ArtifactId, Version**: Define o identificador do projeto.
- **Properties**: Especifica a versão do Java utilizada (Java 17).

2. Dependências

- **JAXB (Java Architecture for XML Binding)**
  - Permite o processamento de dados em formato XML, essencial para integração com sistemas legados que utilizam XML.
  - jaxb-api: Define a API padrão.
  - jaxb-runtime: Implementação de tempo de execução para JAXB.
- **JWT (JSON Web Token)**
  - io.jsonwebtoken:jjwt: Responsável pela criação e validação de tokens JWT, que são amplamente utilizados para autenticação baseada em token.
  - Versão recomendada: 0.9.1, mas pode-se considerar uma versão mais recente, dependendo do suporte necessário.
- **OpenAPI (Springdoc OpenAPI)**
  - org.springdoc:springdoc-openapi-ui: Gera documentação interativa da API (Swagger UI) de forma automática.
  - A documentação Swagger facilita o entendimento da API para desenvolvedores externos e fornece uma interface para testes rápidos.
- **Spring Boot Starter Web**
  - spring-boot-starter-web: Facilita a criação de APIs RESTful e serviços web. Inclui o Tomcat embutido e a dependência para manipulação de JSON.
- **Spring Boot DevTools**
  - spring-boot-devtools: Fornece ferramentas para desenvolvimento, incluindo atualização automática de mudanças (hot reload) e desativação de caches para testes em desenvolvimento.
- **Spring Boot Starter Test**
  - spring-boot-starter-test: Inclui bibliotecas para realizar testes unitários e de integração. Suporta JUnit, AssertJ e Mockito.

3. Build e Plugins

- **Spring Boot Maven Plugin**
  - spring-boot-maven-plugin: Plugin necessário para empacotar a aplicação como um JAR executável, o que permite a execução

autônoma da aplicação. Também facilita a execução de tarefas de build, como empacotamento e publicação de JARs.

-

Chamada Postman

1) Gera Token

POST

Endpoint: http://localhost:8080/educaciencia/token/login

Request

```
{
   "username": "user123",
   "password": "password123"
}
```

Response

```
{
   "message": "HMAC EDUCACIENCIA",
   "Bearer ":
"eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ1c2VyMTIzIiwiaWF0IjoxNzMwNDAzMTQ1LCJleHAiOjE3MzA0MDMyNjV9.4dHDonAzSbI6TZT1J-NZFf_2ElAQacpDcGZv_L5th64"
}
```

---

2) API ML

POST

Endpoint: http://localhost:8080/api/machinelearning/educaciencia/classifica

Bearer Token

Request

```
{
  "analise": "Produtos e serviços novos e inovadores projetados para o futuro continuam a confiar no Java também.Embora a maioria dos aplicativos Java modernos combine o runtime e o aplicativo Java, ainda há muitos aplicativos e até mesmo alguns sites que só funcionarão se você tiver um Java instalado no desktop."
}
```

Response

```
{
   "classificacao": "Level_5"
}
```

Token expirado – após 2 minutos

```
{
   "error": "Token inválido ou expirado."
}
```

Chamada Java

Chama_API_ML.java

```
package api.machinelearning;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.net.HttpURLConnection;
import java.net.URL;
import java.nio.charset.StandardCharsets;

public class Chama_API_ML {

	public static String classifyText(String apiUrl, String token, String analysisRequest) {
		try {

			URL url = new URL(apiUrl + "/api/machinelearning/educaciencia/classifica");
			HttpURLConnection conn = (HttpURLConnection) url.openConnection();

			conn.setRequestMethod("POST");
			conn.setRequestProperty("Content-Type", "application/json");
			conn.setRequestProperty("Authorization", "Bearer " + token);
			conn.setDoOutput(true);

			String jsonInputString = "{\"analise\":\"" + analysisRequest + "\"}";

			try (OutputStream os = conn.getOutputStream()) {
				byte[] input = jsonInputString.getBytes(StandardCharsets.UTF_8);
				os.write(input, 0, input.length);
			}

			int responseCode = conn.getResponseCode();
			if (responseCode == HttpURLConnection.HTTP_OK) {
				BufferedReader in = new BufferedReader(new
InputStreamReader(conn.getInputStream()));
				StringBuilder response = new StringBuilder();
				String line;
				while ((line = in.readLine()) != null) {
					response.append(line);
				}
				in.close();

				return response.toString();
			} else {
				System.out.println("Erro na classificação: Código de resposta HTTP " + responseCode);
			}
		} catch (Exception e) {
			e.printStackTrace();
		}
		return null;
	}

	public static void main(String[] args) {

		String apiUrl = "http://localhost:8080";
		LoginRequest loginRequest = new LoginRequest("user123", "password123");
		String token = GeraToken.getAuthToken(apiUrl, loginRequest);

		if (token != null) {
			System.out.println("*****************************************");
			String analysisRequest = "ODM é excelente";
			String response = classifyText(apiUrl, token, analysisRequest);
```

```
                System.out.println("Resposta da classificação: " + response);
            } else {
                System.out.println("Falha ao obter o token.");
            }
            System.out.println("*****************************************");
        }
}
```

Output

```
*****************************************
******** GERA TOKEN **********************
Resposta da API: {"message":"HMAC EDUCACIENCIA","Bearer
":"eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ1c2VyMTIzIiwiaWF0IjoxNzMwNDA1NDE5LCJleHAiOjE3MzA0MDU1Mzl9.oZU
-jNLxrxBhljMW1Gt36Xv9lTDsyr3ihm7Fi4x7FdE"}
*****************************************
Resposta da classificação: {"classificacao":"Level_5"}
*****************************************
```

Conclusão

A implementação da API com autenticação HMAC e classificação de texto por Machine Learning demonstrou a capacidade de unir segurança avançada e análise automatizada de dados em um único projeto.
Com o uso do Spring Boot, a API oferece uma estrutura escalável e segura, integrando validações de autenticação JWT e HMAC para proteção contra acessos não autorizados.
A análise de texto, proporcionada pelo módulo de Machine Learning, traz inteligência aos dados recebidos, fornecendo insights valiosos para aplicações que necessitam de categorização em tempo real.
Este projeto exemplifica uma solução completa para autenticação e análise de dados, refletindo as melhores práticas em segurança e inteligência artificial.
Ele serve como uma base sólida para desenvolvedores que buscam criar APIs seguras e inteligentes, capazes de atender às demandas modernas de segurança e análise de dados.

EducaCiência FastCode para a comunidade