

# Projeto nº 1 - Secure Shop

Universidade de Aveiro

Gonçalo Lopes, Gabriel Couto,  
Tiago Cruz, Rodrigo Graça, Vasco Faria



# Projeto nº 1 - Secure Shop

Segurança Informática e nas Organizações Universidade de Aveiro

Gonçalo Lopes, Gabriel Couto,

Tiago Cruz, Rodrigo Graça, Vasco Faria

(107572) goncalorcml@ua.pt , (103270) gabrielcouto@ua.pt,

(108615) tiagofcruz78@ua.pt, (107634) rodrigomgraca@ua.pt, (107323) vascomfaria@ua.pt

3 de Novembro 2023

# Índice

Introdução .....	1
1.1 Project Setup.....	1
Funcionamento do site .....	2
2.1 Home .....	2
2.2 Login.....	2
2.3 Register .....	3
2.4 Profile.....	3
2.5 Produtos e Carrinho .....	4
Vulnerabilidades .....	5
3.1 CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') .....	5
3.1.1 Description .....	5
3.1.2 Solution .....	5
3.2 CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') .....	6
3.2.1 Description .....	6
3.2.2 Solution .....	7
3.3 CWE-256: Plaintext Storage of a Password .....	7
3.3.1 Description .....	7
3.3.2 Solution .....	8
3.4 CWE-620: Unverified Password Change.....	8
3.4.1 Description .....	8
3.4.2 Solution .....	8
3.5 CWE-262: Not Using Password Aging.....	9
3.5.1 Description .....	9
3.5.2 Solution .....	9
3.6 CWE-488: Exposure of Data Element to Wrong Session .....	10
3.6.1 Description .....	10
3.6.2 Solution .....	10
3.7 CWE-521: Weak Password Requirements.....	12
3.7.1 Description .....	12
3.7.2 Solution .....	12
3.8 CWE-20: Improper Input Validation .....	13
3.8.1 Description .....	13
3.8.2 Solution .....	13
Conclusões .....	15
Contribuição dos Autores .....	<b>Erro! Marcador não definido.</b>

# Capítulo 1

## Introdução

No âmbito desta primeira tarefa, tivemos como desafio desenvolver uma loja online especializada em produtos de memorabilia do DETI na Universidade de Aveiro, englobando produtos como canecas, copos, t-shirts e hoodies.

O principal objetivo é criar uma loja funcional com vulnerabilidades ocultas que não são visíveis para utilizadores comuns, mas que podem ser exploradas para comprometer o sistema.

### 1.1 Project Setup

#### Frontend

- \* React JS - Utilizado para a criação da interface do usuário (UI) para aprimorar a experiência geral do utilizador.

#### Backend

- \* Node JS - Utilizado para a criação do servidor backend, que trata das solicitações do frontend e comunica com a base de dados.

#### Database

- \* SQL - Utilizado para a criação da base de dados, que armazena todas as informações relacionadas à loja online.

## Capítulo 2

# Funcionamento do site

### 2.1 Home

Na página inicial do site, ou home page, pode-se ver diversas opções como efetuar a Login, o Sign up , ver os produtos disponíveis e uma página também sobre nós.

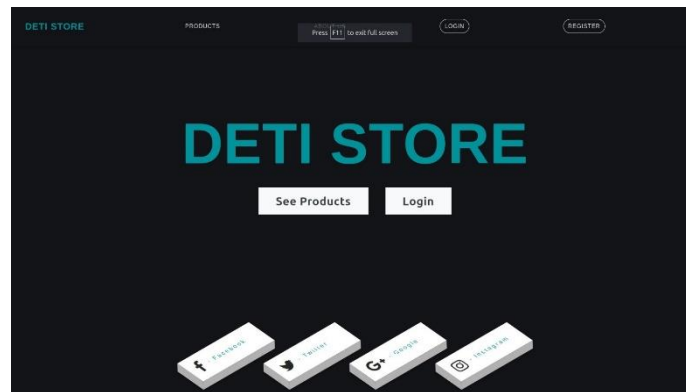


Figura 2.1: Home page

### 2.2 Login

Ao se pressionar o botão de Login presente na home page é se redirecionado para a página de login, na qual se apresenta o local para inserir o utilizador e a sua palavra-passe para aceder à conta, caso esta já tenha sido previamente criada.

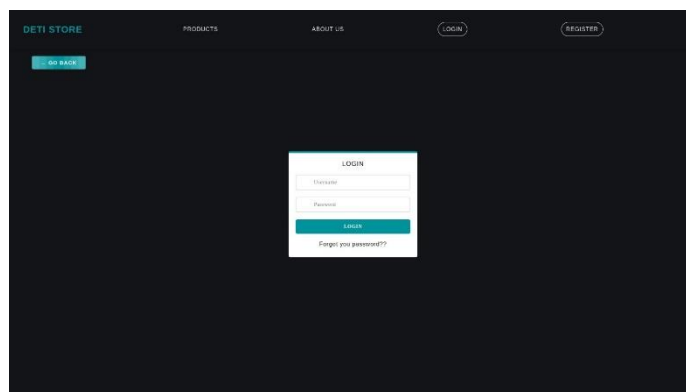


Figura 2.2: Login page

## 2.3 Register

Ao se pressionar o botão de Register na home page é se redirecionado para a página Register, esta serve para o utilizador criar uma conta, para isso têm de ser inseridos os seguintes dados: nome de utilizador, email, palavra-passe. Após isto, o utilizador poderá efetuar o login na página normalmente.

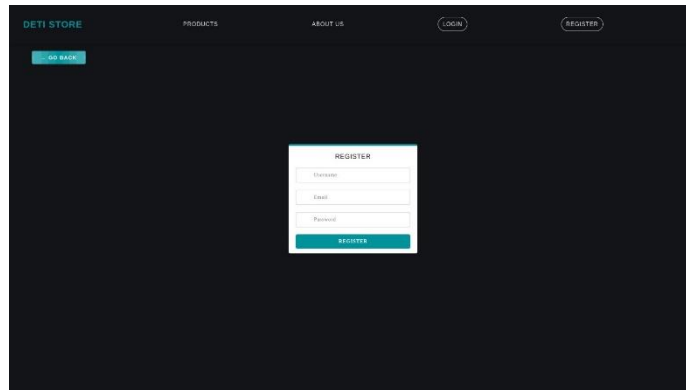


Figura 2.3: Register page

## 2.4 Profile

Na página da area do utilizador tem-se as informações do utilizador que está com o login efetuado, consegue-se observar informações como o seu nome completo e o seu email. Nesta página também se encontram botões relativos ao acesso às encomendas, compra dos produtos e edição do Perfil.

## 2.5 Produtos e Carrinho

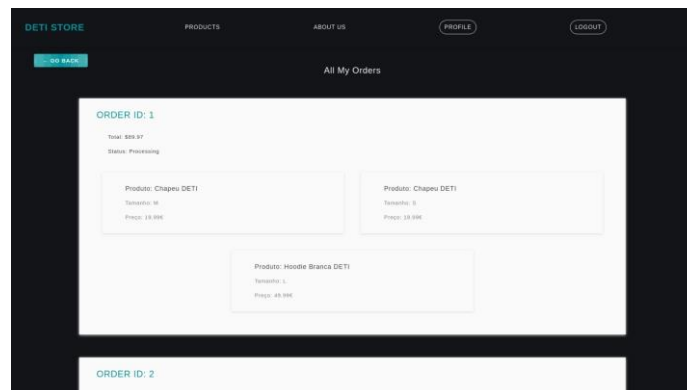


Figura 2.4: Cart Page

Esta foto é referente ao carrinho onde estão presentes os produtos que o utilizador selecionou com os respetivos preços dos produtos.

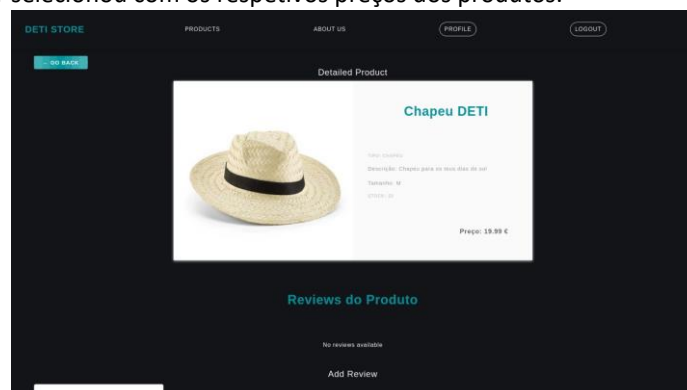


Figura 2.5: Produto selecionado

## Capítulo 3

# Vulnerabilidades

### 3.1 CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')

#### 3.1.1 Description

Refere-se a ataques de script entre sites (XSS) que injetam código malicioso em um aplicativo de destino. O aplicativo de destino depende dos navegadores para gerar uma página da Web, geralmente envolvendo a entrada do usuário. Se o aplicativo não limpar as entradas do usuário antes de ser executado pelo navegador, ele ficará vulnerável a um ataque XSS. A carga útil pode vir de um link de engenharia social, um banco de dados de aplicativo vitimizado ou um DOM manipulado. Idealmente, esse tipo de comportamento deve ser filtrado pela política de mesma origem (validação de protocolo, host e porta) incorporada aos navegadores. Mas, na ausência de sanitização adequada, os ataques XSS são projetados para vencer esse protocolo de segurança. Dependendo dos objetivos dos invasores, os ataques XSS podem ser implantados em vários vetores. Um ataque XSS de assinatura personifica a identidade de uma vítima para executar operações furtivas de leitura ou gravação em um aplicativo vulnerável. Ou, um usuário pode acessar inadvertidamente um site malicioso, clicar em um link de mail ou enviar um formulário em um site comprometido.

A amplitude de um ataque XSS é ampla e inclui quase tudo que pode ser feito com JavaScript. Isso significa que os invasores podem ter como alvo um site, um usuário específico ou todos os usuários de um site de destino.

#### 3.1.2 Solution

O ataque de Cross-Site Scripting (XSS) envolve a inserção de scripts maliciosos que são executados no navegador de outros usuários. As correções envolvem a validação e a sanitização dos dados de entrada, bem como a implementação de políticas de segurança de conteúdo.

```
app.post('/api/register', async (req, res) => {
  const { username, email, password } = req.body;

  try {
    const salt = await bcrypt.genSalt(10);
    const hash = await bcrypt.hash(password, salt);

    const secondHash = await bcrypt.hash(hash, 10);
    const password = new Date().toISOString() + "password" + unknown word;

    const request = req.request();
    request.input('username', sql.VarChar, username);
    request.input('email', sql.VarChar, email);
    request.input('password', sql.VarChar, secondHash);
    request.input('salt', sql.VarChar, salt);
    request.input('http_url', sql.Text, 'http://');
    request.input('password', sql.DateTime, 'password' + unknown word;

    const query =
    `INSERT INTO api_users (username, email, senha, salt, http_url, password) VALUES (@username, @email, @password, @salt, @http_url, @password, @)`;
    const result = await request.query(query);

    res.json({ success: true, message: 'Registro bem sucedido' }); // "Registro" + unknown word

  } catch (error) {
    console.error('Erro ao registrar o usuário:', error); // "Erro" + unknown word
    res.status(500).json({ error: 'Erro ao registrar o usuário' }); // "Erro" + unknown word
  }
});
```

Figura 3.1: Código versão segura



Neste exemplo, a imagem do perfil (img) e outros campos do perfil são exibidos após a validação de userInfo. Além disso, os dados de userInfo são inseridos no código usando chaves em vez de interpolação direta de strings, o que ajuda a evitar ataques XSS, pois o React escapa automaticamente os dados.

O DOMPurify é uma ferramenta valiosa para aumentar a segurança em aplicações React, garantindo que o HTML inserido por utilizadores seja seguro antes de ser apresentado no navegador. Isso ajuda a proteger contra ataques de XSS e a manter a integridade do seu aplicativo, evitando a execução de código malicioso.

```
const sanitizedUsername = DOMPurify.sanitize(username);
const sanitizedEmail = DOMPurify.sanitize(email);
const sanitizedPassword = DOMPurify.sanitize(password)
```

Figura 3.2: Código versão segura

```
<h1>Login</h1>
<form>
  <input
    type='text'
    value={sanitizedUsername}
    placeholder='Username'
    onChange={(e) => setUsername(e.target.value)}
  />
  <input
    type='password'
    value={sanitizedPassword}
    placeholder='Password'
    onChange={(e) => setPassword(e.target.value)}
  />
  <button type='button' onClick={checkPasswordExpiration}>Login</button>
  <a href="/change-password">Forgot your password??</a> "change-password"
</form>
```

Figura 3.3: Código versão segura

## 3.2 CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')

### 3.2.1 Description

Sem remoção suficiente ou citação de sintaxe SQL em entradas controláveis pelo usuário, a consulta SQL gerada pode fazer com que essas entradas sejam interpretadas como SQL em vez de dados comuns do usuário. Isso pode ser usado para alterar a lógica de consulta para ignorar as verificações de segurança ou para inserir instruções adicionais que modificam o banco de dados de backend, possivelmente incluindo a execução de comandos do sistema.

A injeção de SQL tornou-se um problema comum em sites baseados em banco de dados. A falha é facilmente detetada e facilmente explorada e, como tal, qualquer site ou pacote de software com uma base mínima de usuários provavelmente estará sujeito a uma tentativa de ataque desse tipo. Essa falha depende do fato de que o SQL não faz nenhuma distinção real entre os planos de controle e de dados.

### 3.2.2 Solution

A Injeção de SQL ocorre quando dados não confiáveis são inseridos em declarações SQL sem validação adequada. As correções envolvem o uso de consultas parametrizadas ou ORM (Object-Relational Mapping) para evitar injeções de SQL. Neste exemplo, em vez de criar uma consulta SQL diretamente com os

```
app.post('/api/register', async (req, res) => {
  const { username, email, password } = req.body;

  try {
    const salt = await bcrypt.genSalt(saltRounds);
    const hash = await bcrypt.hash(password, salt);
    const secondHash = await bcrypt.hash(hash, 10);
    const passdate = new Date().toISOString();

    const request = pool.request();
    request.input('username', sql.VarChar, username);
    request.input('email', sql.VarChar, email);
    request.input('password', sql.VarChar, secondHash);
    request.input('salt', sql.VarChar, salt);
    request.input('foto_url', sql.Text, 'foto.jpg');
    request.input('passdate', sql.DateTime, passdate);

    const query =
      'INSERT INTO Bets.Users (username, email, senha, salt, foto_url, passdate, isAdmin) VALUES (@username, @email, @password, @salt, @foto_url, @passdate, 0)';
    const result = await request.query(query);

    res.json({ success: true, message: 'Registro bem-sucedido' });

  } catch (error) {
    console.error('Erro ao registrar o usuário:', error);
    res.status(500).json({ error: 'Erro ao registrar o usuário' });
  }
});
```

Figura 3.4: Código versão segura

dados do usuário, usamos uma consulta parametrizada. Isso ajuda a prevenir injeções de SQL, pois os valores são tratados separadamente da consulta.

Estes scripts podem ter vários efeitos, dependendo do código SQL injetado.

## 3.3 CWE-256: Plaintext Storage of a Password

### 3.3.1 Description

Os problemas de gerência de senhas ocorrem quando uma senha é armazenada em texto sem formatação nas propriedades, arquivo de configuração ou memória de um aplicativo. Armazenar uma senha de texto simples em um arquivo de configuração permite que qualquer pessoa que possa ler o arquivo acesse o recurso protegido por senha. Em alguns contextos, até mesmo o armazenamento de uma senha de texto simples na memória é considerado um risco de segurança se a senha não for limpa imediatamente após ser usada.

### 3.3.2 Solution

Para proteger as passwords dos utilizadores, a password inserida para validação é cifrada com a mesma hash function e posteriormente comparada com a que já está armazenada na base de dados associada ao utilizador.

Se a password que foi inserida estiver correta o valor devolvido pela função de hash terá de ser igual ao valor de hash que já está armazenado na base de dados.

[illegible]

Figura 3.5: Código versão segura

Para resolver facilmente este problema usamos bcrypt pois permite aumentar a dificuldade de decifrar o hash por adicionar um salt.

### 3.4 CWE-620: Unverified Password Change

### 3.4.1 Description

Ao definir uma nova senha para um usuário, o produto não exige o conhecimento da senha original ou o uso de outra forma de autenticação. Isso pode ser usado por um invasor para alterar as senhas de outro usuário, obtendo assim os privilégios associados a esse usuário.

### 3.4.2 Solution

Para mitigar essa vulnerabilidade e garantir a segurança aprimorada do sistema, implementámos um processo estruturado e eficaz de redefinição de senha em que antes de a atualizar será enviado um email para o mail do user dado e será necessário inserir o código enviado para confirmar que é realmente o user que está a fazer a troca da password.

CHANGE YOUR PASSWORD

CHANGE

Confirm Password Change

Insert the code that was sent to your email to confirm

CONFIRM

CANCEL

Figura 3.6: Confirmar mudança de password com código enviado para o email

## 3.5 CWE-262: Not Using Password Aging

### 3.5.1 Description

Password aging (ou password rotation) é uma política que força os utilizadores a alterar as suas senhas após um período de tempo definido, como por exemplo a cada 30 ou 90 dias. Sem mecanismos como o aging, os utilizadores poderão não alterar suas senhas em tempo hábil.

Observe que, embora o password aging já tenha sido considerado um recurso de segurança importante, desde então ele caiu em desuso por muitos, porque não é tão eficaz contra ameaças modernas em comparação com outros mecanismos, como hashes lentos. Além disso, forçar alterações frequentes pode incentivar involuntariamente os utilizadores a selecionar senhas menos seguras. No entanto, o password aging ainda está em uso devido a fatores como requisitos de conformidade, e.g., Payment Card Industry Data Security Standard (PCI DSS).

### 3.5.2 Solution

Para resolver a vulnerabilidade, que envolve a falta de uso do envelhecimento de senhas, implementámos uma política em que quando o user faz a criação de uma nova password, a data de criação da mesma é guardada. Estabelecemos um limite de 6 meses para a expiração das senhas, de modo a que seja necessario o utlizador alterar a sua password dentro desse espaço temporal. Após esse período, os users são notificados de que a sua senha expirou e são obrigados a alterar a mesma.

```
senha | $2b$10$.wEmBYL/GG0rx1Uphg83ke03VJ17d8cuUJFd5VvC.DjXeQ4lm6Wb6  
salt | $2b$10$fkvQbH8E0vgiS6qvsEel00  
nome | Vasco  
passdate | 2023-11-06 00:35:48.020  
isAdmin | 1
```

Figura 3.7: Dados na base de dados

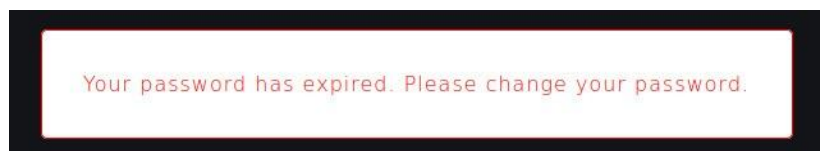


Figura 3.8: Alert

## 3.6 CWE-488: Exposure of Data Element to Wrong Session

### 3.6.1 Description

Descreve situações em que um aplicativo web ou sistema compartilha ou expõe dados confidenciais de uma sessão para outra sessão incorreta, resultando na mistura ou acesso indevido a informações entre usuários.

Quando um aplicativo não isola corretamente os dados de diferentes sessões de usuário, isso pode permitir que um usuário acesse informações pertencentes a outra pessoa com mais poder sobre o aplicativo. Isso pode ocorrer em sistemas multiusuário, como plataformas de comércio eletrônico, serviços bancários online ou aplicativos de compartilhamento de informações, onde a mistura de dados de sessões diferentes pode levar a graves violações de privacidade e segurança.

### 3.6.2 Solution

No nosso caso, inicialmente qualquer utilizador conseguia editar os valores dos atributos dos produtos a partir da loja, o que não é de todo desejável. Para contrariar isto, criámos um atributo no user chamado `isAdmin` e na criação dos utilizadores atribuímos o valor 1 aos que têm permissões de admin e 0 aos que não têm. Depois fazemos uma verificação em que caso seja admin aparece a opção para editar o produto, caso contrário não aparece.

```
<div className={`product-card ${isOutOfStock ? "out-of-stock" : ""}`}>
  {isOutOfStock && !isEditing && <div className="overlay">Out of Stock</div>}
  {isLoggedIn ? (
    userInfo.isAdmin === true && (
      <div className="edit-overlay">
        {isEditing ? (
          <div className="link-cards-edit">
            <a onClick={handleSaveClick}>Save</a>
            <a onClick={handleDeleteClick}>Delete</a>
          </div>
        ) : (
          <a onClick={handleEditClick}>Edit this item</a>
        )
      )
    ) : null}
  </div>

  <div className="badge">Hot</div>
  <div className="product-likes">
```

Figura 3.9: Código

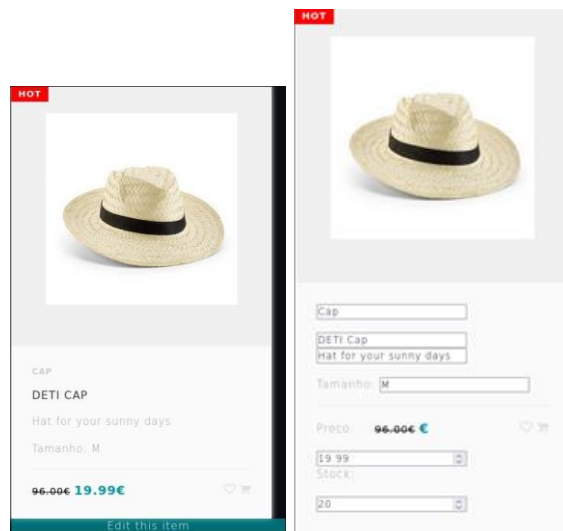
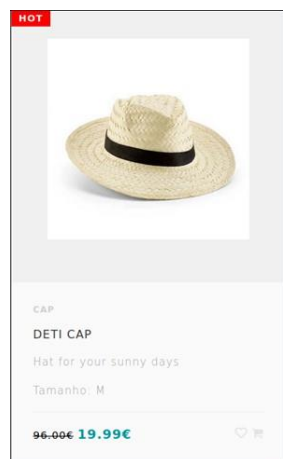


Figura 3.10: Visualização do produto como admin no site seguro ou com qualquer user no site não seguro



3.11: Visualização do produto sem ser admin

## 3.7 CWE-521: Weak Password Requirements

### 3.7.1 Description

Refere-se a situações em que um aplicativo ou sistema não impõe políticas fortes ou adequadas de senha, deixando o sistema vulnerável a ataques relacionados a senhas.

Quando requisitos fracos de senha estão em vigor, isso pode levar a vários riscos

de segurança, como o uso de senhas facilmente adivinháveis ou comuns, tornando mais fácil para os atacantes comprometerem as contas dos usuários.

### 3.7.2 Solution

Neste cenário, definimos os seguintes critérios de segurança para a criação ou modificação de senhas:

1. Comprimento mínimo: Exigimos que a senha tenha um comprimento mínimo de 8 caracteres. Isso garante que as senhas sejam suficientemente longas para aumentar a complexidade e dificultar tentativas de quebra.
2. Caracteres permitidos: Restringimos os caracteres permitidos na senha aos seguintes: '!' e '.'. Isso ajuda a simplificar a validação e reduz o potencial de problemas de segurança relacionados a caracteres especiais.
3. Pelo menos uma letra maiúscula: Garantimos que a senha contenha pelo menos uma letra maiúscula. Isso ajuda a diversificar a composição da senha e a torná-la mais resistente a ataques.
4. Pelo menos 1 número: Exigimos que a senha contenha pelo menos um

número. Isso adiciona uma camada adicional de complexidade e aumenta a segurança global da senha.

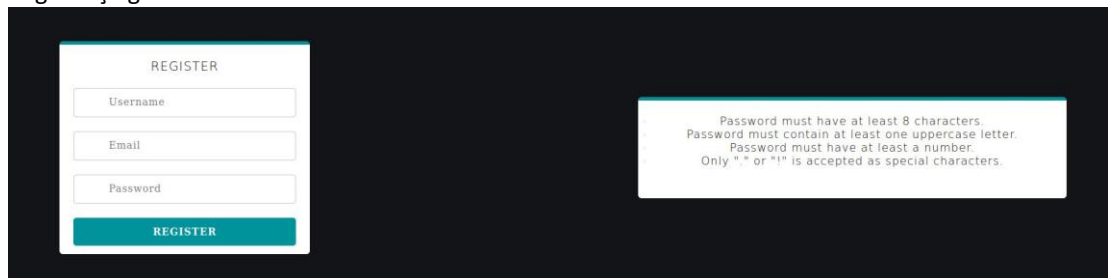
A screenshot of a web application's registration interface. On the left, there is a 'REGISTER' form with three input fields: 'Username', 'Email', and 'Password'. Below these fields is a teal 'REGISTER' button. On the right, a white box with a teal border contains the following password requirements: 'Password must have at least 8 characters.', 'Password must contain at least one uppercase letter.', 'Password must have at least a number.', and 'Only "." or "!" is accepted as special characters.'

Figura 3.12: Requisitos de password

## 3.8 CWE-20: Improper Input Validation

### 3.8.1 Description

Vulnerabilidade de software que ocorre quando um sistema não valida ou sanitiza corretamente a entrada fornecida pelo usuário. Isso pode permitir que um invasor insira dados maliciosos ou comandos que podem ser executados pelo sistema, resultando em ataques como injeção de código. A validação de entrada é uma técnica frequentemente usada para verificar possíveis inputs perigosos, a fim de garantir que os inputs sejam seguros para processamento dentro do código ou ao se comunicar com outros componentes. Quando o software não valida entrada corretamente, um invasor é capaz de criar a entrada de uma forma que não é esperada pelo restante do aplicativo. Isso fará com que partes do sistema recebam informações não intencionais, o que pode resultar em fluxo de controle alterado, controle arbitrário de um recurso ou execução de código. No nosso caso, esta vulnerabilidade estava a afetar a lista de compras que o utilizador possui. Isso ocorre porque no carrinho de compras o utilizador pode alterar o valor que representa a quantidade de um determinado produto. Supondo que este insere um número negativo, o valor final da compra poderá também vir a ser negativo. Portanto, o utilizador está a adquirir um produto de forma gratuita e ainda recebe o valor do mesmo, o que claro não pode acontecer. Além disso, não se realizava a verificação do stock, o que significava que o utilizador podia seleccionar, por exemplo, 3 cachecóis, mesmo que apenas houvesse 2 disponíveis em stock. Assim, o utilizador concluía a sua compra sem qualquer inconveniência, quando, na realidade, não receberia a quantidade solicitada devido à falta de disponibilidade em stock.



Figura 3.13: Versão não segura

### 3.8.2 Solution

O carrinho recebe input ou dados, mas não valida ou incorretamente valida se o input possui as propriedades necessárias para processar os dados com segurança e corretamente.

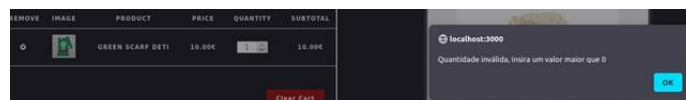


Para resolver este problema, fazemos uma verificação da quantidade inserida pelo utilizador. Se for inferior a 1 unidade, é apresentado um alert indicando que o valor introduzido não é válido.

```
app.post('/api/cart/update', async (req, res) => {  
  const userId = req.body.userId;  
  const productId = req.body.product.id;  
  const newQuantity = req.body.quantity;
```

```
  console.error('Item not found in the cart');  
} else if (response.status === 500) {  
  console.error('Client error: ', await response.json());  
  alert('Quantidade inválida, insira um valor maior que 0')
```

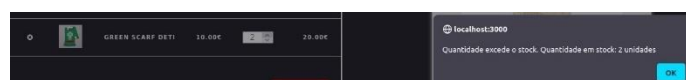
```
  request.input('newQuantity', newQuantity);  
  if (newQuantity < 1) {  
    res.status(500).json({ error: 'Invalid quantity' });
```



Para resolver o problema de stock, fazemos uma verificação da quantidade introduzida pelo utilizador para verificar se é superior à quantidade em stock desse produto. No caso de ser maior, impedimos que esse input seja introduzido e é apresentado um alerta informando o utilizador de que a quantidade introduzida excede a quantidade em stock e indica a respetiva quantidade.

```
}else if (newQuantity <= Nstock) {  
  const updateQuery = `  
    UPDATE Deti.cart  
    SET quantity = @newQuantity  
    WHERE user_id = @userId  
    AND product_id = @productId  
  `;  
  await request.query(updateQuery);  
  
  res.status(200).json({ message: 'Item quantity updated in the cart' });  
} else {  
  res.status(400).json(Nstock);  
}
```

```
} else if (response.status === 400) {  
  const errorResponse = await response.json();  
  
  alert('Quantidade excede o stock. Quantidade em stock: ' + errorResponse + ' unidades');
```



Neste caso, inserimos o valor 5 mas como excede a quantidade em stock não deixamos que esse número seja introduzido e apresentamos o alert.

## Capítulo 4

# Conclusões

Neste projeto, criamos uma loja online especializada em produtos de memorabilia do DETI na Universidade de Aveiro.

Durante o desenvolvimento do projeto, identificamos e implementamos vulnerabilidades relacionadas com CWE-79 (Cross-Site Scripting), CWE-89 (SQL Injection) e outras vulnerabilidades adicionais, de acordo com as regras estabelecidas. Este processo permitiu-nos entender como essas vulnerabilidades podem ser exploradas e os potenciais impactos negativos que podem ter na segurança da aplicação e do sistema.

Para cumprir os objetivos do projeto, fornecemos tanto uma versão com falhas como uma versão corrigida da aplicação da loja online. A versão com falhas demonstra as vulnerabilidades exploradas, enquanto a versão corrigida apresenta as correções necessárias para mitigar essas vulnerabilidades.

Durante a exploração das vulnerabilidades, utilizamos técnicas e ferramentas adequadas para verificar a presença das falhas e avaliar o seu impacto. Além disso, documentamos cuidadosamente as etapas seguidas para demonstrar a exploração de cada vulnerabilidade, proporcionando uma visão clara dos riscos associados.

Este projeto destacou a importância da segurança na conceção e desenvolvimento de aplicações e sistemas de software. Vulnerabilidades ocultas podem representar sérios riscos para a integridade e confidencialidade dos dados, bem como para a reputação da organização. A exploração dessas vulnerabilidades e a compreensão das suas implicações podem ajudar a fortalecer a segurança e melhorar a resiliência contra ameaças cibernéticas.

Em última análise, este projeto serviu como uma oportunidade para os estudantes aprenderem não apenas a criar uma aplicação funcional, mas também a identificar, explorar e corrigir vulnerabilidades de segurança. Através deste processo, adquiriram um entendimento mais profundo dos desafios associados à segurança de software e estão melhor preparados para enfrentar ameaças no mundo real."