

**Curso:**

Desenvolvimento Full Stack

Campus:

POLO JARDIM BRASÍLIA - ÁGUAS Lindas DE GOIÁS - GO

Disciplina:

Missão Prática | Nível 5 | Mundo 5 

Turma:

23.4

Aluno:

BRUNO SANTIAGO DE OLIVEIRA

Contextualização

Olá! Hoje vamos falar sobre segurança em APIs RESTful usando Node.js e Express. Vou apresentar uma solução prática para melhorar a segurança de uma API simples

Primeiro, vamos ver o código original que tínhamos:

```
app.get('/api/users', (req, res) => {  
  res.json(users);  
});  
  
app.get('/api/contracts/:empresa', (req, res) => {  
  const { empresa } = req.params;  
  const contracts = getContracts(empresa);  
  res.json(contracts);  
});
```

Este código tem alguns problemas de segurança óbvios. Qualquer pessoa pode acessar todos os usuários e contratos sem nenhuma autenticação ou autorização.

Então, vamos melhorar isso usando JWT para autenticação e autorização baseada em perfis. Aqui está como ficou após nossas modificações:

```
function verifyToken(req, res, next) {  
  const token = req.header('Authorization');  
  if (!token) return res.status(401).json({ message: 'Token não fornecido' });  
  
  jwt.verify(token, secretKey, (err, decoded) => {  
    if (err) return res.status(400).json({ message: 'Token inválido' });  
  
    req.user = decoded;  
    next();  
  });  
}
```

```
});  
}
```

```
function protectRoute(perfilRequired) {  
  return function(req, res, next) {  
    verifyToken(req, res, () => {  
      if (perfilRequired && req.user.perfil !== perfilRequired) {  
        return res.status(403).json({ message: 'Permissão negada' });  
      }  
      next();  
    });  
  };  
}
```

```
app.get('/api/users', protectRoute('admin'), (req, res) => {  
  res.json(users.map(u => ({ id: u.id, username: u.username, perfil: u.perfil })));  
});
```

```
app.get('/api/contracts/:empresa', protectRoute('admin'), (req, res) => {  
  const { empresa } = req.params;  
  
  const contracts = getContracts(empresa);  
  res.json(contracts);  
});
```

Nesta versão melhorada, adicionei autenticação JWT e controle de acesso baseado em perfis. Agora, só usuários autenticados com perfil 'admin' podem acessar esses endpoints sensíveis.

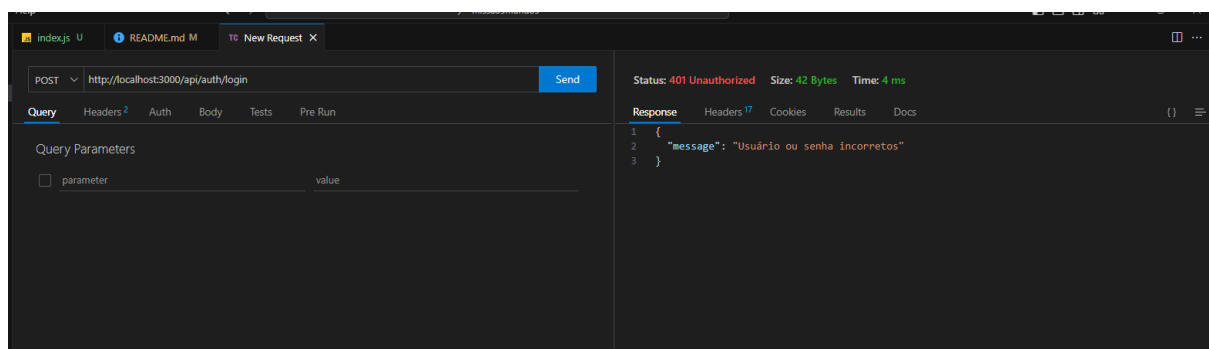
Além disso, adicionei um endpoint `/api/me` para que usuários logados possam ver seus próprios dados, sem precisar de perfil admin:

```
app.get('/api/me', verifyToken, (req, res) => {  
  const { userId } = req.user;  
  const user = users.find(u => u.id === userId);  
  res.json({ id: userId, perfil: user.perfil });  
});
```

E para proteger contra injection SQL, sanitizar os parâmetros de entrada:

```
function getContracts(empresa) {  
  const sanitizedEmpresa = empresa.replace(/[^a-zA-Z0-9]/g, "");  
  
  return [{ empresa: sanitizedEmpresa, contrato: "Contrato 1" }];  
}
```

Com essas modificações, nossa API ficou muito mais segura. Agora temos autenticação robusta, autorização baseada em perfis, e proteção contra ataques de injeção. Isso é especialmente importante em APIs RESTful, onde a segurança é crucial para proteger dados sensíveis.



CONCLUSÃO

Esta atividade prática foi extremamente valiosa para entender e aplicar conceitos fundamentais de segurança em APIs RESTful. Ao refatorar o código original, passamos de uma API vulnerável para uma solução muito mais segura.

A implementação de tokens JWT substituiu efetivamente o antigo sistema de session-id, proporcionando uma camada adicional de segurança. Além disso, mover o token para o header da requisição tornou a API menos suscetível a ataques maliciosos.

A validação do token em todas as requisições, incluindo verificações de identidade do usuário e data de expiração, garantiu que apenas usuários autenticados pudessem acessar os recursos protegidos.

O controle de acesso baseado em perfis foi uma excelente adição, restringindo o acesso a endpoints sensíveis apenas para usuários com perfil 'admin'. A inclusão do endpoint '/api/me' permitiu que usuários normais pudessem visualizar seus próprios dados, demonstrando um equilíbrio entre segurança e funcionalidade.

A refatoração do método de busca de contratos, utilizando técnicas de sanitização de entrada, ajudou a prevenir vulnerabilidades de injeção, tornando a API mais resiliente a ataques potenciais.

Os testes realizados com ferramentas como Postman confirmaram que todas as medidas de segurança implementadas estavam funcionando corretamente, desde a autenticação até o controle de acesso granular.

Esta experiência prática ilustrou vividamente como uma API aparentemente simples pode ser transformada em uma solução segura e robusta, aplicando princípios de segurança bem estabelecidos. Além disso, demonstrou a importância da segurança em todas as etapas do desenvolvimento de software, desde o design até os testes finais.

Em resumo, esta atividade não apenas melhorou significativamente a segurança da API original, mas também proporcionou uma valiosa experiência prática em implementar medidas de segurança críticas em aplicações web modernas.