

**Engenharia de Software Seguro**

# **Relatório - Laboratórios 1 e 2**

Sistema multiutilizador de gestão de listas de tarefas

Grupo 6

Elementos: Hugo Nunes, Miguel Teixeira

Repositório GitHub: [c-academy-ess/api-todo-list-manager-grupo-6](https://github.com/c-academy-ess/api-todo-list-manager-grupo-6)

Tag(s): lab1, lab2

Data: 06-02-2026

## Sumário

1. Laboratório 1 - Ferramentas, requisitos e primeira versão da API
2. Laboratório 1 - Base de dados e análise de segurança
3. Laboratório 2 - HTTPS, autenticação (PBKDF2 + JWT) e autorização (RBAC)
4. Laboratório 2 - OAuth2/OIDC com Google Tasks
5. Conclusões

## 1. Laboratório 1 - Ferramentas, requisitos e primeira versão da API

### 1.1 Objetivos

O Laboratório 1 teve como objetivo levantar requisitos para um sistema multiutilizador de listas de tarefas e iniciar a implementação do back-end (API, serviços e repositórios), primeiro com persistência em memória e, depois, com persistência relacional.

### 1.2 Ferramentas e ambiente

- IntelliJ IDEA Community Edition + plugin PlantUML (diagramas UML).
- Maven e JDK 21 (compilação/execução fora do IDE).
- Docker Desktop (execução do PostgreSQL).
- Postman (testes à API) e pgAdmin 4 (gestão da base de dados).

### 1.3 Criação do repositório GitHub

O repositório do grupo foi criado via GitHub Classroom. O desenvolvimento foi marcado com tags para cada entrega.

### 1.4 Histórias de utilizador

Foram consideradas as histórias fornecidas no enunciado e acrescentadas as seguintes:

#### 1.4.1 História de partilha de listas

Como utilizador autenticado (dono de uma lista), quero partilhar uma lista com outros utilizadores para que possam consultar as tarefas dessa lista.

Critérios de aceitação:

- O sistema deve permitir introduzir o e-mail de outro utilizador registado para partilha.
- O utilizador convidado deve conseguir ver a lista na sua área pessoal após a partilha.
- Apenas o dono da lista deve ter permissão para remover o acesso a outros utilizadores.

#### 1.4.2 História de segurança (CRUD de tarefas)

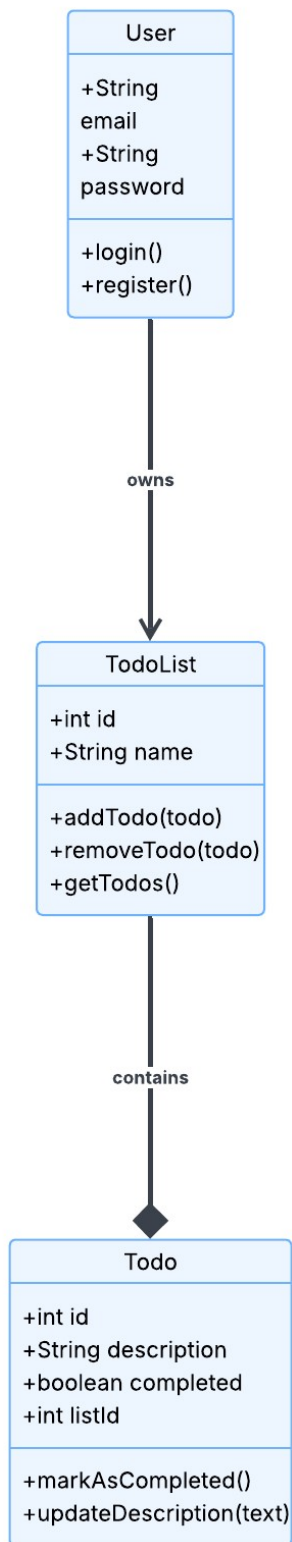
Como utilizador autenticado, quero que o sistema **verifique as minhas permissões** sempre que tento criar, ver, editar ou remover uma tarefa, para garantir que apenas eu ou utilizadores autorizados possamos manipular os meus dados.

- **Princípio Aplicado:** Deve seguir o princípio do **Privilégio Mínimo**, garantindo que um utilizador regular não tem acesso às tarefas de outros sem autorização expressa.
- **Controlo:** A aplicação deve **Autorizar após Autenticar**, validando se o utilizador é efetivamente o dono da lista antes de permitir qualquer alteração (conforme o fluxo da API do laboratório).

### 1.5 Diagrama de classes

O diagrama de classes das entidades principais (User, TodoList e Todo) pode ser representado com PlantUML como segue:

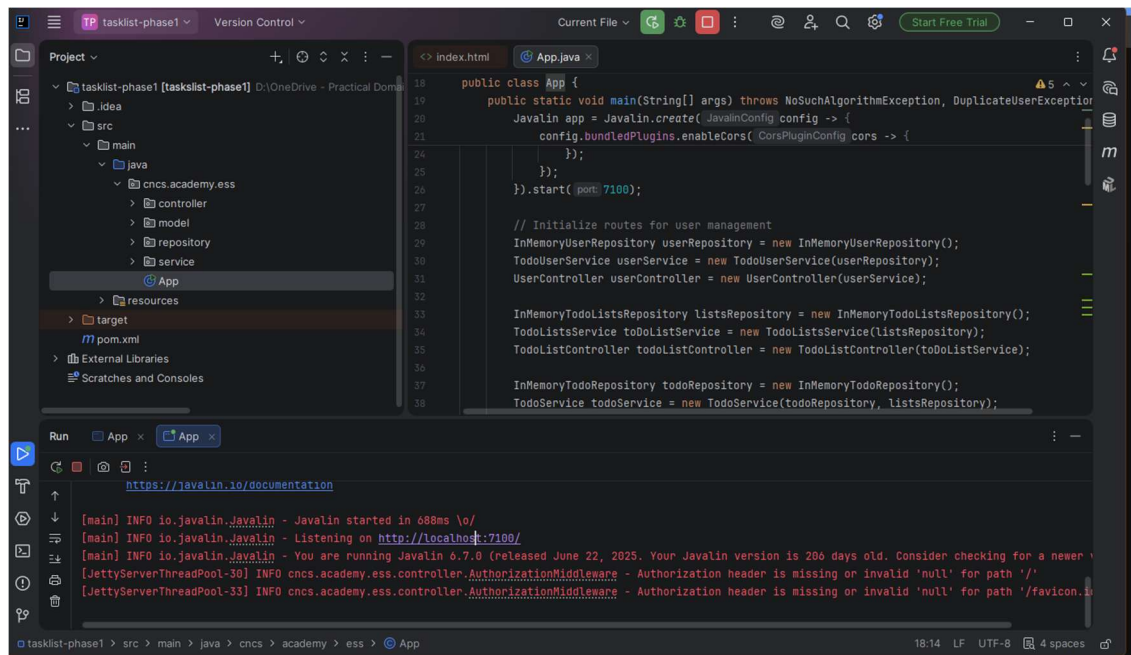
# Todo App System



## 1.6 Execução da versão em memória (tasklist-phase1) e testes da API

A aplicação foi executada localmente na porta 7100 e testada via Postman. O fluxo típico de testes foi:

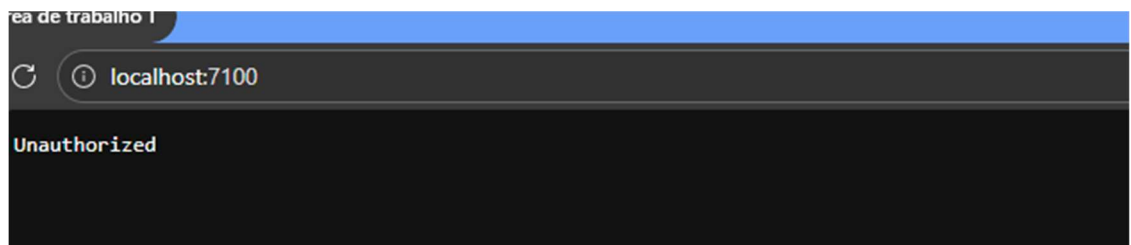
- Criar utilizador (POST /user)
- Login (POST /login) para obter token
- Criar lista (POST /todolist) usando Authorization: Bearer <token>
- Criar tarefa (POST /todo/item) e listar tarefas (GET /todo/{listId}/tasks)



```
18 public class App {
19     public static void main(String[] args) throws NoSuchAlgorithmException, DuplicateUserException {
20         Javalin app = Javalin.create(JavalinConfig config -> {
21             config.bundledPlugins.enableCors(CorsPluginConfig cors -> {
22                 cors.allowOrigin("http://localhost:7100");
23             });
24         });
25         app.start(port: 7100);
26
27         // Initialize routes for user management
28         InMemoryUserRepository userRepository = new InMemoryUserRepository();
29         TodoUserService userService = new TodoUserService(userRepository);
30         UserController userController = new UserController(userService);
31
32         InMemoryTodoListsRepository listsRepository = new InMemoryTodoListsRepository();
33         TodoListsService todoListsService = new TodoListsService(listsRepository);
34         TodoListController todoListController = new TodoListController(todoListsService);
35
36         InMemoryTodoRepository todoRepository = new InMemoryTodoRepository();
37         TodoService todoService = new TodoService(todoRepository, listsRepository);
38     }
```

Run console output:

```
[main] INFO io.javalin.Javalin - Javalin started in 688ms \o/
[main] INFO io.javalin.Javalin - Listening on http://localhost:7100/
[main] INFO io.javalin.Javalin - You are running Javalin 6.7.0 (released June 22, 2025. Your Javalin version is 286 days old. Consider checking for a newer version.)
[JettyServerThreadPool-30] INFO cnacs.academy.ess.controller.AuthorizationMiddleware - Authorization header is missing or invalid 'null' for path '/'
[JettyServerThreadPool-33] INFO cnacs.academy.ess.controller.AuthorizationMiddleware - Authorization header is missing or invalid 'null' for path '/favicon.ico'
```



ase1



... / Create User



Save



Share



POST



localhost:7100/user

Send



Body



Cookies

raw



JSON



Schema

Beautify

```
1 {  
2   "username": "useress",  
3   "password": "cncs**2024"  
4 }
```

Body



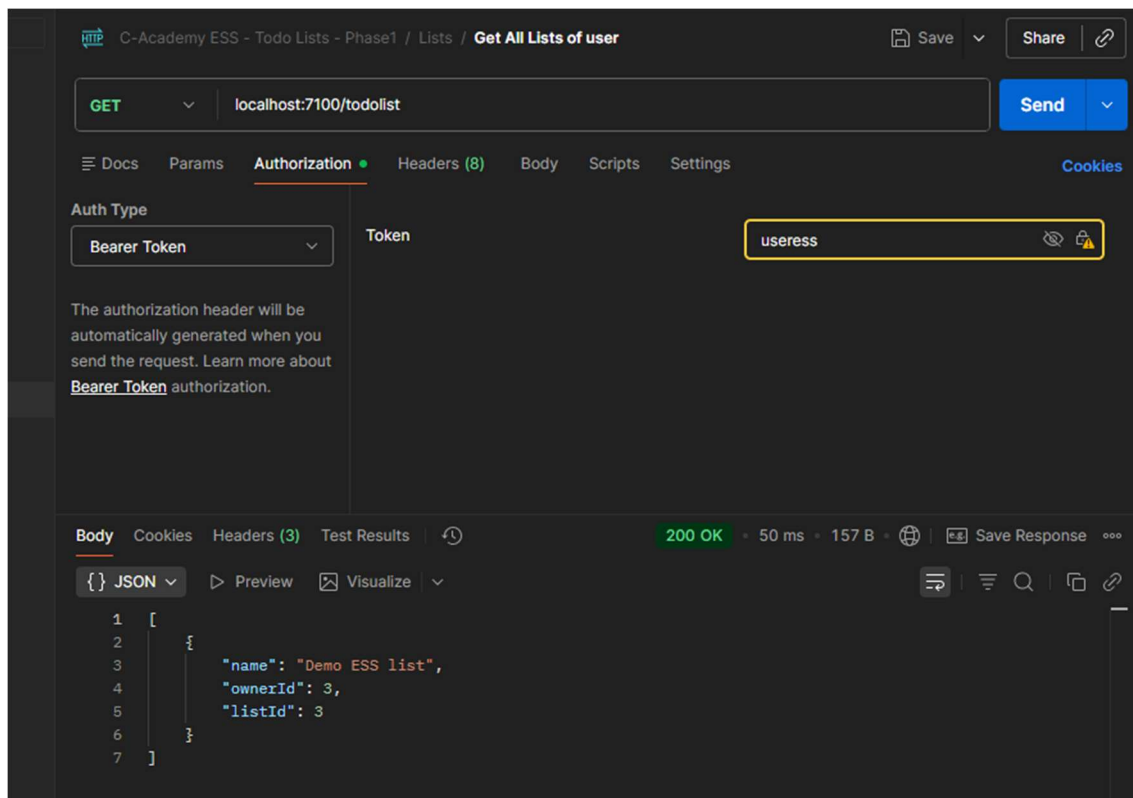
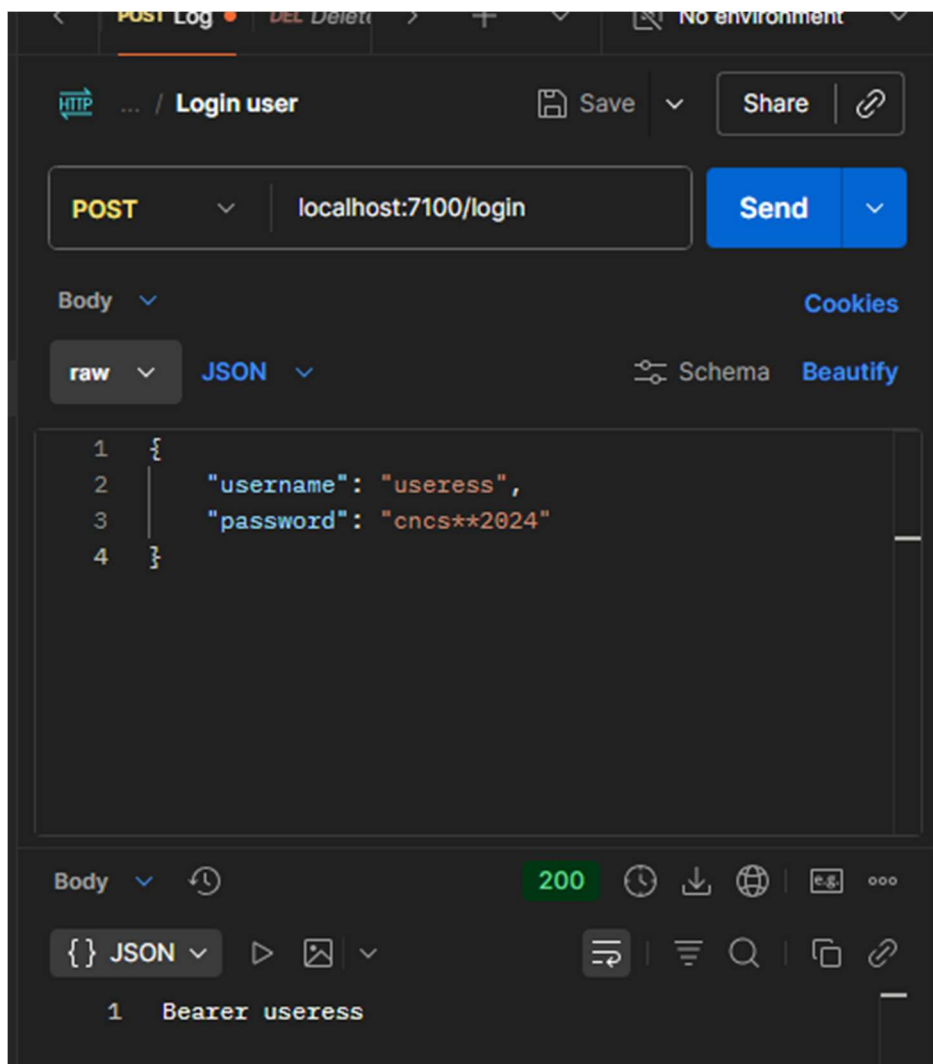
201



{ } JSON



```
1 {  
2   "userId": 3,  
3   "username": "useress"  
4 }
```



## 1.7 Inconsistência identificada: verificação de dono da lista

A verificação de que a lista pertence ao utilizador autenticado deveria ser realizada na camada de Service, e não no Controller. O Controller deve apenas tratar do pedido HTTP e delegar a lógica de negócio e controlo de acesso específico ao Service, garantindo melhor separação de responsabilidades.

## 1.8 Funcionamento do AuthorizationMiddleware (fase 1)

A classe **AuthorizationMiddleware** é responsável por garantir que apenas utilizadores **autenticados** conseguem aceder às rotas protegidas da API.

**Funcionamento:**

### Interceção do pedido HTTP

O middleware é executado **antes** dos controllers, interceptando todos os pedidos às rotas protegidas.

1. **Leitura do cabeçalho Authorization**
2. Verifica se o pedido contém o cabeçalho HTTP:
3. Authorization: Bearer <token>
4. **Validação do token**
  - Extrai o token do cabeçalho
  - Compara-o com os tokens gerados previamente na rota /login
  - Confirma se o token corresponde a um utilizador válido
5. **Pedido autorizado ou rejeitado**
  - Se o token for **válido** → o pedido prossegue para o controller
  - Se o token for **inválido ou inexistente** → devolve erro HTTP **401 Unauthorized**

### Papel na arquitetura

- Implementa **autenticação transversal** (cross-cutting concern)
- Evita repetir código de validação de token em cada controller
- Centraliza o controlo de acesso inicial à API

O AuthorizationMiddleware:

- **apenas valida se o utilizador está autenticado**
- **não verifica permissões específicas**, como:
  - se o utilizador é dono da lista
  - se pode apagar ou editar uma tarefa



## 2. Laboratório 1 - Base de dados e análise de segurança

### 2.1 Inicialização do PostgreSQL e criação do esquema

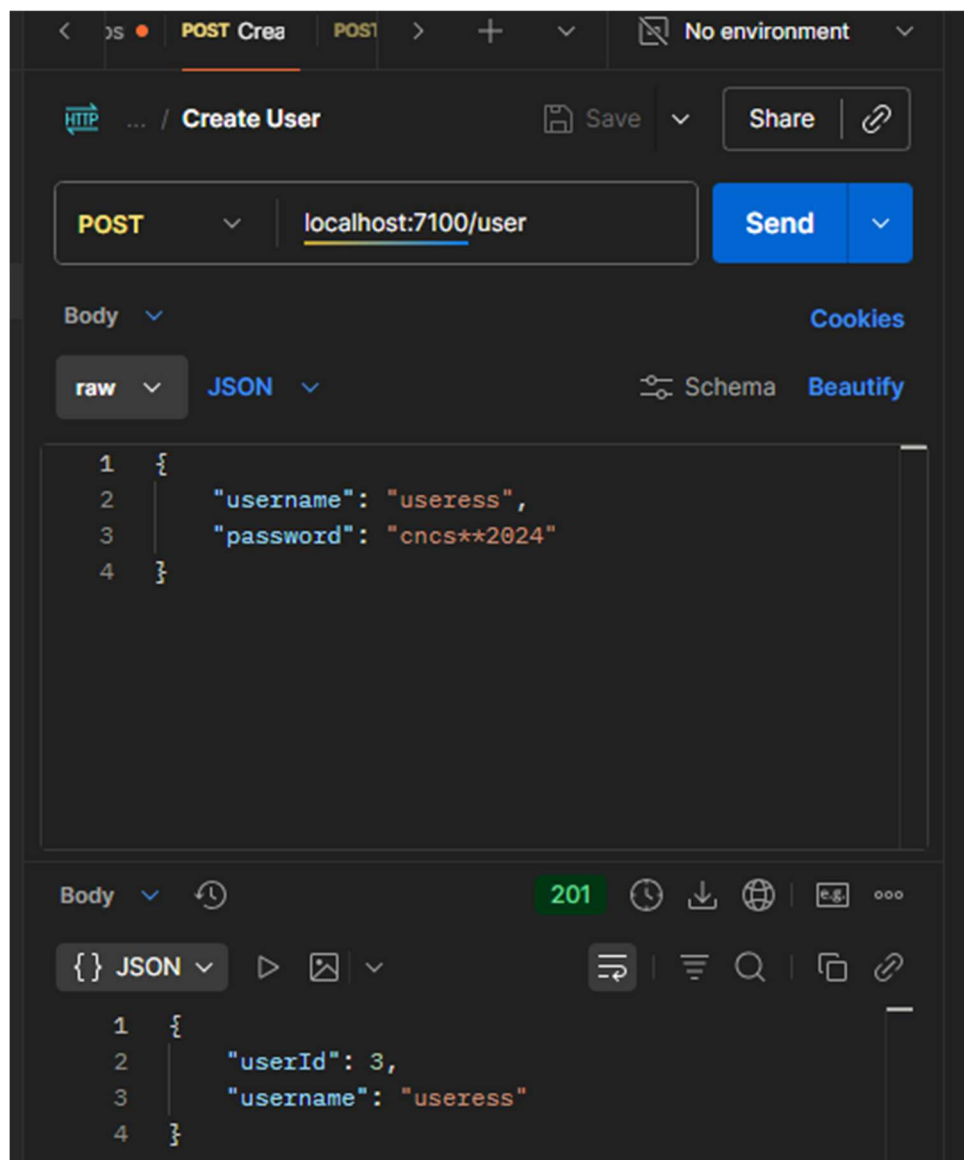
Foi iniciada uma instância PostgreSQL via Docker e criado o esquema de dados executando os scripts `users.sql`, `lists.sql` e `todos.sql`. Comandos e evidências devem ser incluídos abaixo.

```
# Iniciar postgres (sem persistência)

docker run --name postgres-academy -e POSTGRES_PASSWORD=changeit -d -p 5432:5432
postgres
```

### 2.2 Execução da versão com SQL (tasklist-phase2) e repositórios SQL

A versão phase2 liga a aplicação a PostgreSQL. A classe `SQLUserRepository` serve de exemplo para implementar `SQLTodoListsRepository` e `SQLTodoRepository`.



Data Output Messages Notifications				
<div> <div>+</div> <div>▼</div> <div>▼</div> <div>▼</div> <div>▼</div> <div>▼</div> <div>▼</div> <div>▼</div> </div>				
	id	username	password	
	[PK] integer	character varying (255)	character varying (255)	
1	1	user1	password1	
2	2	user2	password2	
3	3	useress	cncs**2024	

Os repositórios `SQLTodoListRepository` e `SQLTodoRepository` foram implementados seguindo exatamente o padrão do `SQLUserRepository`, utilizando `BasicDataSource` e `JDBC`. Cada método obtém a sua própria ligação à base de dados e executa operações SQL para persistência e leitura de listas e tarefas, mantendo a separação entre lógica de negócio e acesso a dados.

## 2.3 Problemas de segurança identificados (Lab1)

### 2.3.1 Credenciais e configuração de BD hardcoded

Na implementação base, as **credenciais da base de dados** (utilizador, password, endereço e porto) estão **escritas diretamente no código fonte**, no ficheiro `app.java`.

#### Riscos

- Exposição acidental das credenciais (GitHub, partilha de código)
- Dificuldade em mudar credenciais entre ambientes (dev/test/prod)
- Violação de boas práticas de segurança (hardcoded secrets)

As credenciais devem ser **externalizadas**, por exemplo:

- Variáveis de ambiente
- Ficheiros de configuração (.properties, .env)
- Ferramentas de gestão de segredos

#### Exemplo (variáveis de ambiente)

- `dataSource.setUrl(System.getenv("DB_URL"));`
- `dataSource.setUsername(System.getenv("DB_USER"));`
- `dataSource.setPassword(System.getenv("DB_PASSWORD"));`

### 2.3.2 Password em claro e token fraco

#### Problema 1: Armazenamento inseguro de passwords

Na implementação base:

- As passwords são armazenadas **em texto simples** ou com **hash fraco**
- Não é usado **salt** nem algoritmos adequados

#### Exploração possível

- Se a base de dados for comprometida, as passwords ficam imediatamente expostas
- Facilita ataques de reutilização de credenciais

#### Correção

- Usar algoritmos de hashing seguros:
  - bcrypt
  - PBKDF2
  - Argon2
- Aplicar **salt automático**

#### Problema 2: Token de autenticação fraco

Na implementação base:

- O token devolvido no /login é previsível (ex.: username)
- Não expira
- Não é assinado

#### Exploração possível

- Um atacante pode **forjar tokens**
- Aceder à API como outro utilizador

---

#### Correções recomendadas

- Usar tokens aleatórios e imprevisíveis
- Associar tokens a sessões com tempo de expiração
- Preferencialmente usar **JWT assinado** ou sessões server-side

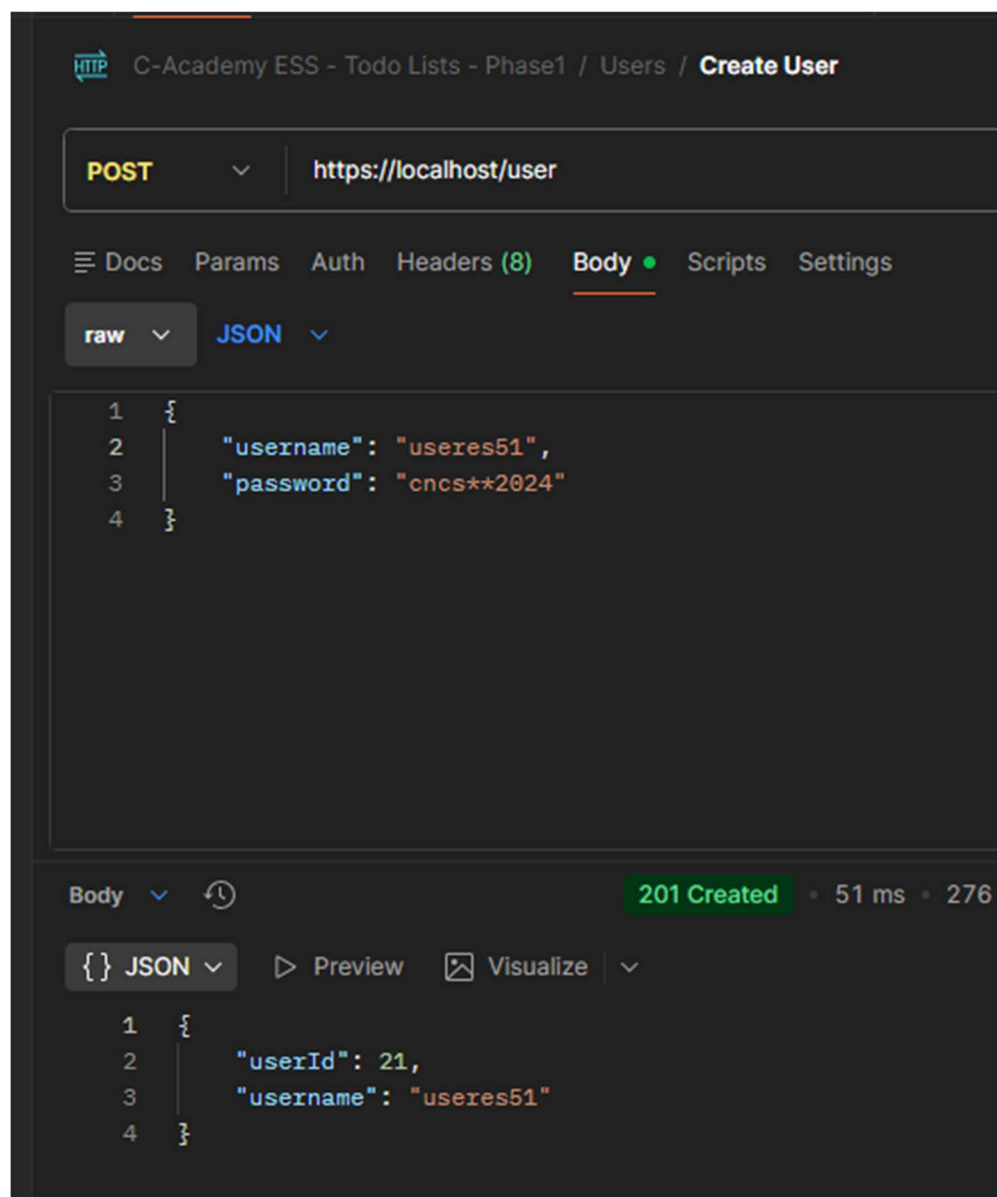
### 3. Laboratório 2 - HTTPS, autenticação (PBKDF2 + JWT) e autorização (RBAC)

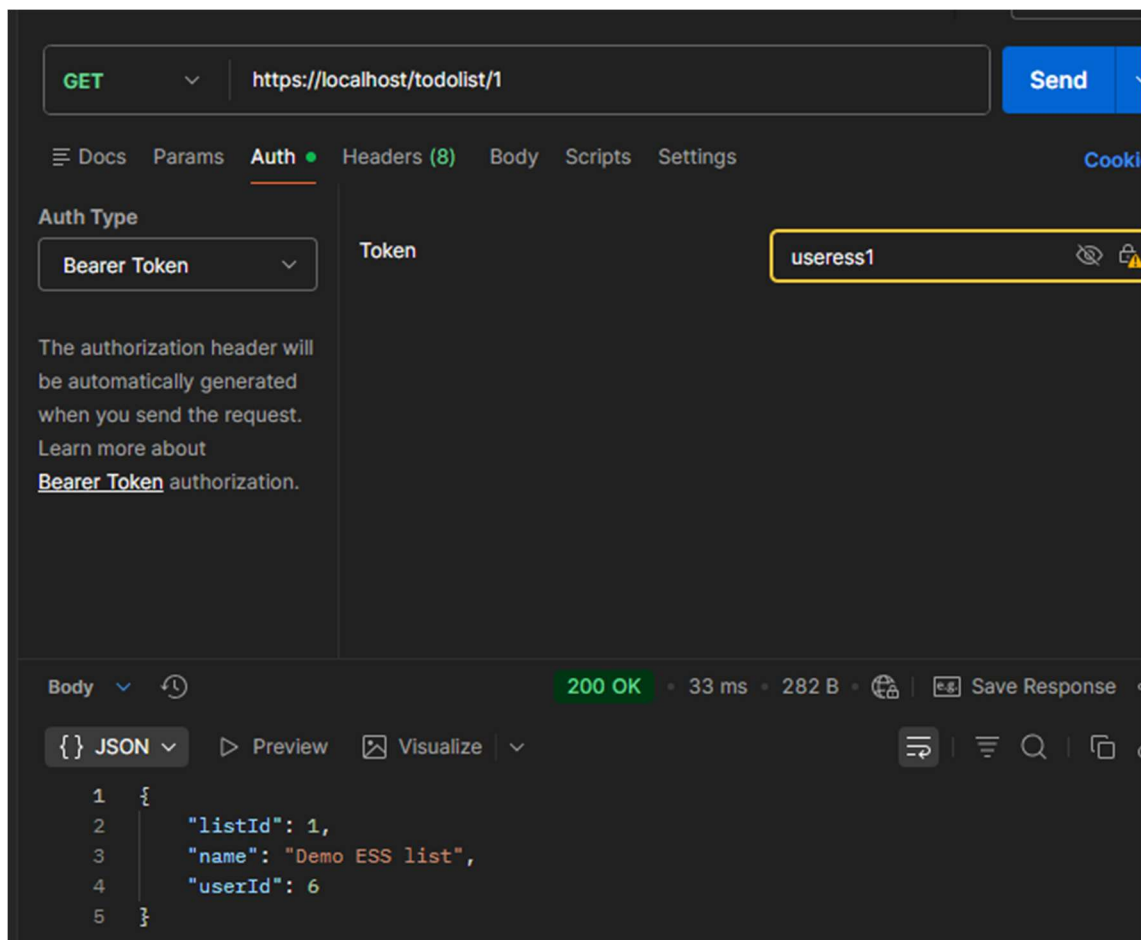
#### 3.1 Configuração de HTTPS (Javalin)

A API foi configurada para aceitar HTTPS com certificado (autoassinado) e chave privada, seguindo o tutorial oficial do Javalin. Em ambiente local, o Postman foi configurado para confiar no certificado (ou para desativar a verificação, apenas para testes).

```
https://javalin.io/documentation

main] INFO io.javalin.Javalin - Javalin started in 699ms \o/
main] INFO io.javalin.Javalin - Listening on http://localhost:80/
main] INFO io.javalin.Javalin - Listening on https://localhost:443/
main] INFO io.javalin.Javalin - You are running Javalin 6.1.4 (released May 4, 2024. Your Javalin version is 6.1.4)
Exception in thread "main" java.lang.RuntimeException: Create breakpoint : Failed to save user
    at cncs.academy.ess.repository.sql.SQLErrorRepository.save(SQLErrorRepository.java:66)
```





### 3.2 Revisão do login: hashing PBKDF2

O login foi reimplementado para guardar apenas um hash PBKDF2 da password (com salt e iterações). No registo, é gerado salt aleatório e armazenado junto do hash; no login, o hash é recomputado e comparado em tempo constante.

```

// Exemplo (simplificado) de geração PBKDF2

SecretKeyFactory skf = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256");

PBEKeySpec spec = new PBEKeySpec(password.toCharArray(), salt, iterations, 256);

byte[] hash = skf.generateSecret(spec).getEncoded();

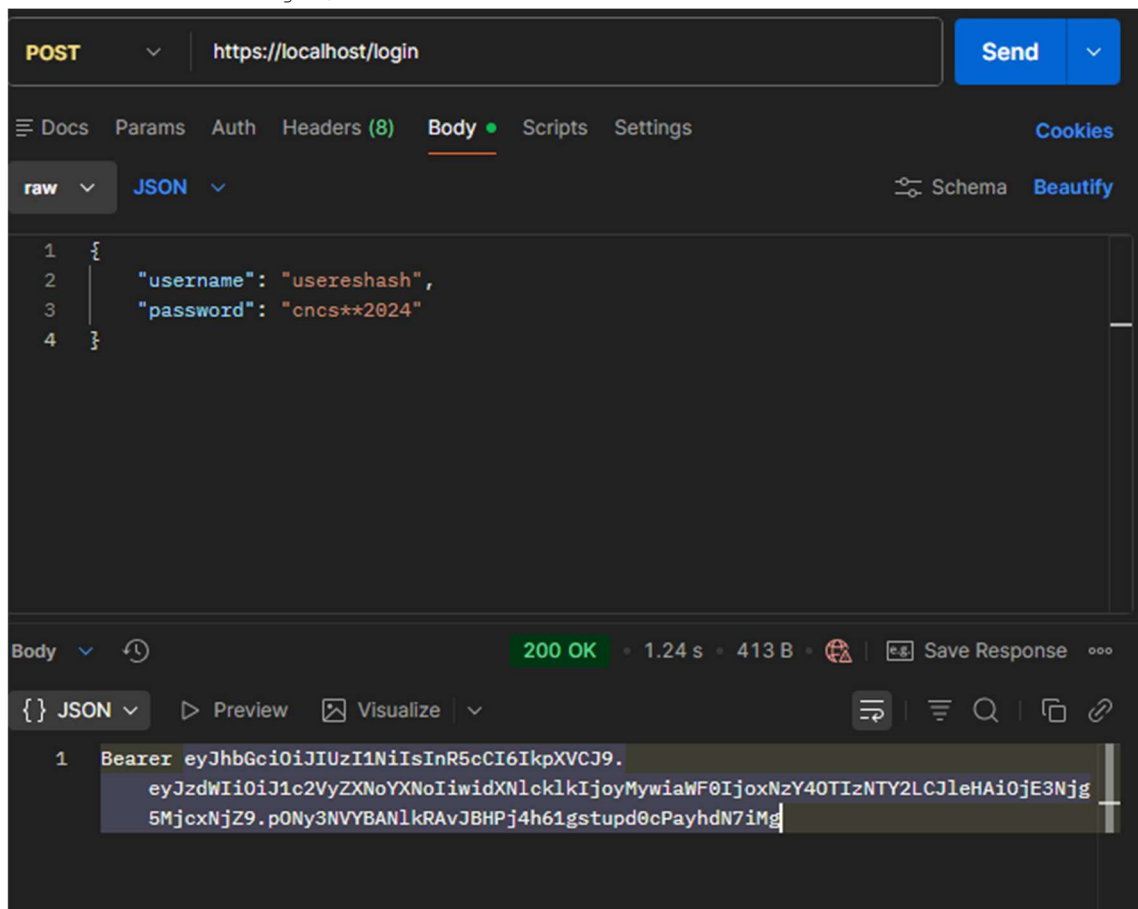
```

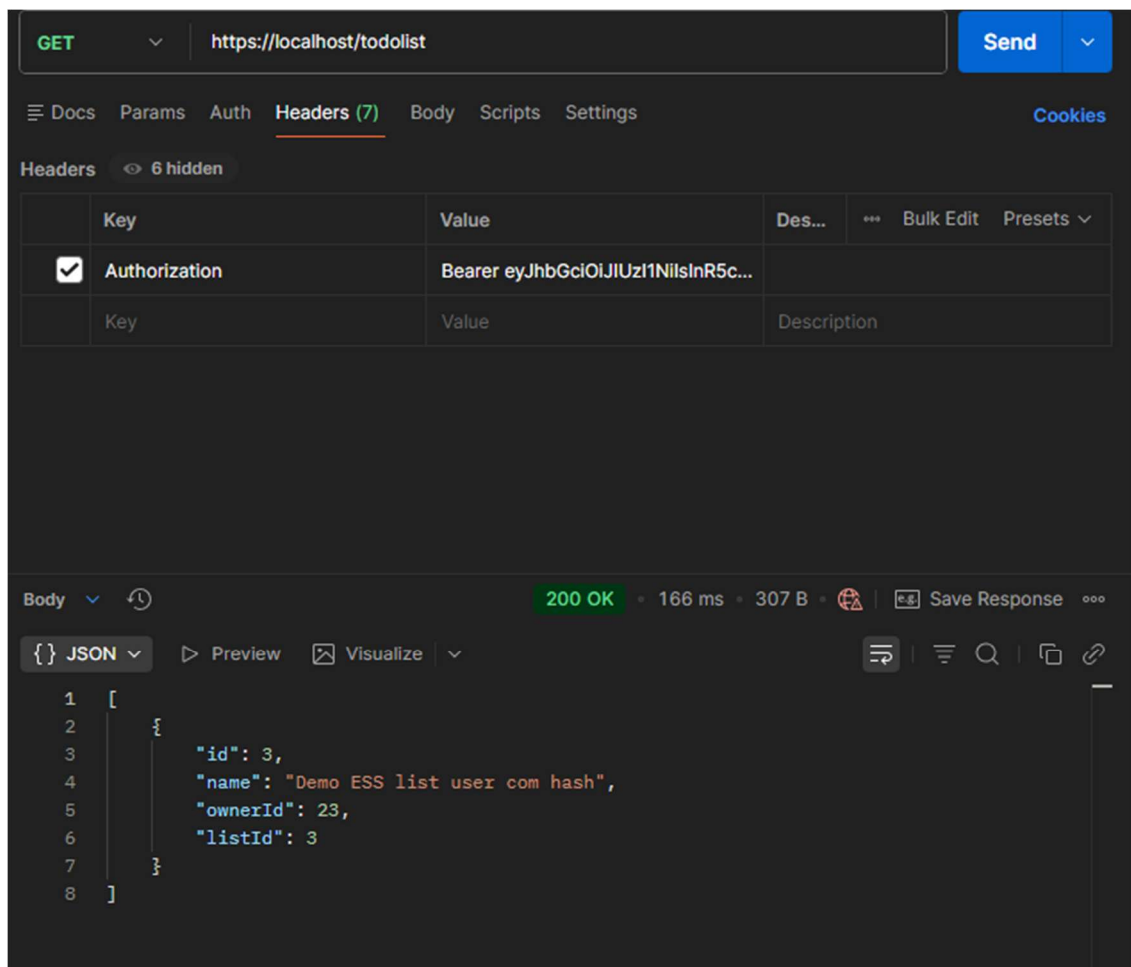
Data Output Messages Notifications			
	id [PK] integer	username character varying (255)	password character varying (255)
1	1	user1	password1
2	2	user2	password2
3	3	useress	cnscs**2024
4	6	useress1	cnscs**2024
5	21	useres51	cnscs**2024
6	23	usereshash	65536:os2FKGA1Euc2imQJJTjqoQ==:ZP3JxLyVFE2/8Xjp0VagTrc4SAFo5FhieUaPLhxX0...

### 3.3 Token JWT assinado

O token de autenticação passou a ser um JWT assinado pelo back-end. Inclui, no mínimo: issuer, username, issuedAt e expiresAt. O middleware valida assinatura e expiração antes de aceitar pedidos.

```
Algorithm alg = Algorithm.HMAC256(secret);
String jwt = JWT.create()
    .withIssuer("todo-api")
    .withClaim("username", username)
    .withIssuedAt(new Date())
    .withExpiresAt(new Date(System.currentTimeMillis() + 3600_000))
    .sign(alg);
return "Bearer " + jwt;
```





Durante os testes iniciais, pedidos autenticados com JWT foram rejeitados enquanto o middleware ainda utilizava o mecanismo de autenticação antigo. Após a adaptação do middleware para validação de tokens JWT assinados, os pedidos autenticados passaram a ser corretamente autorizados, confirmando o funcionamento da solução.

### 3.4 Controlo de acessos (RBAC) com jCasbin

Foi integrado um controlo de acessos RBAC com três roles: Base, Share e Admin. O handler valida (sub, obj, act) para cada rota, usando model.conf e policy.csv carregados em runtime.

#### Model.conf

```
[request_definition]
r = sub, obj, act
[policy_definition]
p = sub, obj, act
[role_definition]
g = _, _
[policy_effect]
e = some(where (p.eft == allow))
[matchers]
m = g(r.sub, p.sub) && r.obj == p.obj && r.act == p.act
```

## Policy.csv

```
# -----  
# BASE role: create lists + tasks (and read)  
# -----  
p, role_base, todoclist, create  
p, role_base, todoclist, read  
  
p, role_base, todoitem, create  
p, role_base, todoitem, read  
p, role_base, todoitem, delete  
  
# -----  
# SHARE role: Base + share lists  
# -----  
p, role_share, todoclist, share  
  
# -----  
# ADMIN role: Share + manage users  
# -----  
p, role_admin, user, create  
p, role_admin, user, delete  
p, role_admin, user, read  
  
# -----  
# Role hierarchy (cumulative permissions)  
# share inherits base; admin inherits share  
# -----  
g, role_share, role_base  
g, role_admin, role_share  
  
# -----  
# Example user-role assignments  
# Sub code ser username OU userId string (ex: "user:1")  
# -----  
g, useress1, role_admin  
g, useress5, role_share  
g, useres51, role_base
```



## Autenticação

- Verifica header Authorization
- Valida JWT assinado
- Rejeita tokens inválidos ou expirados (401)

## ✓ Autorização (RBAC)

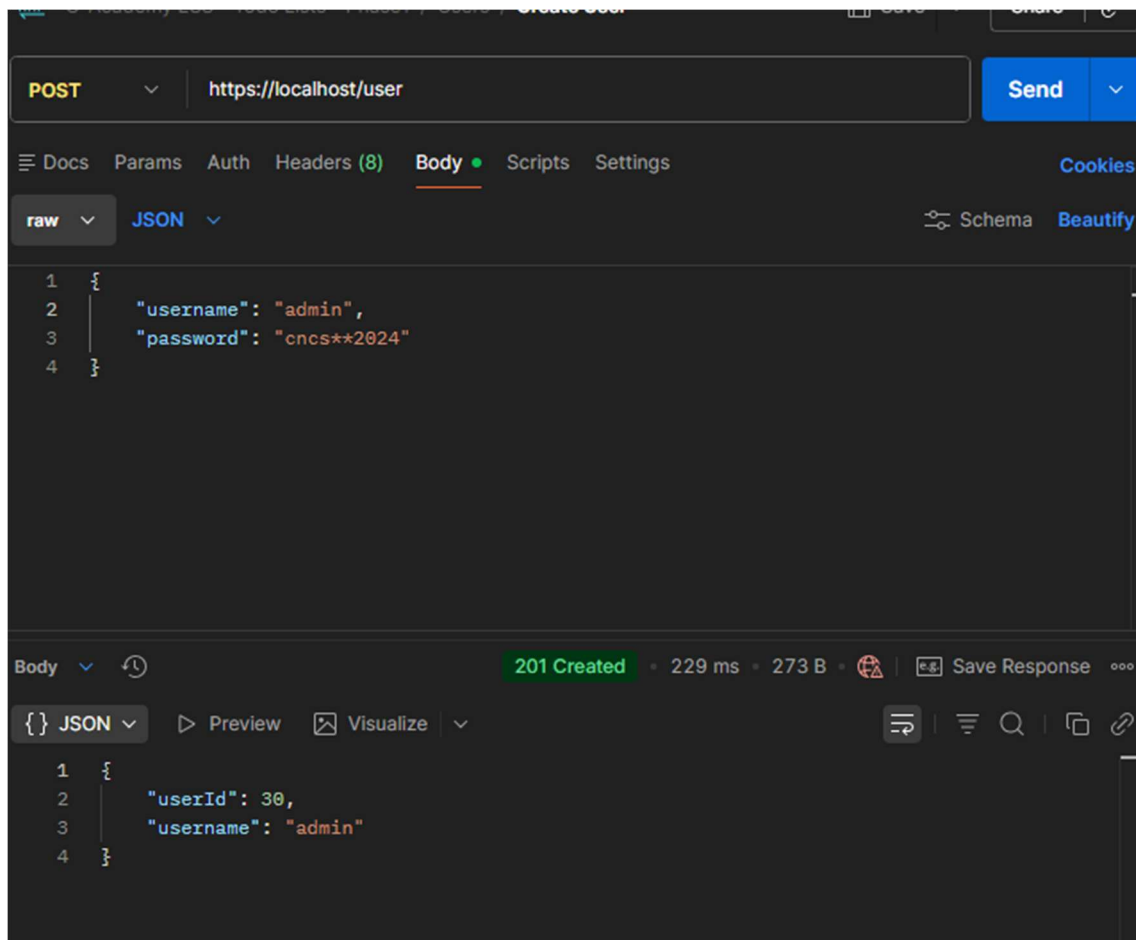
- Diferencia **utilizadores normais** e **admin**
- Aplica **403 Forbidden** quando necessário
- Centraliza regras no middleware

## ✓ Arquitetura correta

- Autenticação ≠ Autorização
- Controllers ficam limpos
- Fácil de evoluir para jCasbin completo

A autorização foi implementada utilizando o modelo RBAC através da biblioteca jCasbin. Após a autenticação via JWT, o papel do utilizador é avaliado e as permissões de acesso são decididas com base em políticas externas definidas em ficheiros de configuração, garantindo controlo de acesso granular e separação entre autenticação e autorização.

```
[main] INFO io.javalin.Javalin - You are running Javalin 6.1.4 (released May 4, 2024. Your Javalin version is 626 days old. Consider checking for a newer version.).
[main] INFO org.casbin.jcasbin - Model:
[main] INFO org.casbin.jcasbin - p.p: sub, obj, act
[main] INFO org.casbin.jcasbin - r.r: sub, obj, act
[main] INFO org.casbin.jcasbin - e.e: some(where (p_eft == allow))
[main] INFO org.casbin.jcasbin - g.g: --
[main] INFO org.casbin.jcasbin - m.m: g(r_sub, p_sub) && keyMatch2(r_obj, p_obj) && regexMatch(r_act, p_act)
[main] INFO org.casbin.jcasbin - Policy:
```



user com role\_base → **403**

user com role\_share → **200**

admin → **200**



Hi ct3ia77@gmail.com

Go back to [Home screen](#)

ENCODED VALUE

Enable auto-focus

JSON WEB TOKEN (JWT)

COPY CLEAR

Valid JWT

Fix public key input errors to verify signature.

eyJhbGciOiJIUzU1NiIsInR5cCI6IkpXLTJlZDkxMjEwLWVudC5ybnBnbiJ9.e3pc3MI0i3hY2NvdW50cySnb29nbGUyY29tIiwiaXpwIjoibDMzMDEMTgzNTc4LWQwODM3dGVIZGkwZWNoIDkNGS4MnRtdDI4YW9wcHISLmFwcmMuZ29vZ2xiZXNlcmlNbvnRlbnQuY29tIiwiaXVkiOiJoNDM0ODEMTgzNTc4LWQwODM3dGVIZGkwZWNoIDkNGS4MnRtdDI4YW9wcHISLmFwcmMuZ29vZ2xiZXNlcmlNbvnRlbnQuY29tIiwic3ViOiJMTExOTCBjIiwiaXNjcXNzcyOOTE4ZDkxZWVudC5ybnBnbiJ9.jdnPpTIC3QgdtYmLSlmNvbStisImVTYmLSX3ZlcmIaaWVKIjpbcnvllCjhdF9oYXNoIjoiciTBX001RkRar3ZnUEYTMCIufukoQSiSlmhdcI6MTc20TE4MJU2MlwiZXhwIjoxNDY5MTg2MTYyYQ..Ydl1siZ5OLKJR3cpE7SDsxAZNkgGGkl-937rSFvmgKIa489RSxYSVMtwzxSURjU8ghzL63bhbcB8FSOAusim90TVxwgITWiSi-jpuPlxtRTUIwI0p8s-928aB278aIx4ysX\_1iY0\_68wgWKSgubfq7roqYVhdSGSTrtN27-67fIJToA-Z4pBU4tg8\_8lk9cpVpMU0Trz4Goh9ePNlp1PhsoykZashC\_Iz94gf1-7gdCyveloJNBt-MfutZ-75SKYS2bwISxbxy6qXjkmCcMQozudiIQ68htd2IKLTd7nsfJ38sQEYKSOIZEAQZugLfYjaAoexokQ5\_SgtQMsluiQw

DECODED HEADER

JSON CLAIMS TABLE

COPY ↗

```
{  
  "alg": "RS256",  
  "kid": "9544dbff0f59b0fe25301643f3275bf68776765822",  
  "typ": "JWT"  
}
```

DECODED PAYLOAD

JSON CLAIMS TABLE

COPY ↗

```
{  
  "iss": "accounts.google.com",  
  "azp": "43681183578-d4q07ebdi0dsi89d4k82smu28aoapp9.apps.googleusercontent.com",  
  "aud": "43681183578-d4q07ebdi0dsi89d4k82smu28aoapp9.apps.googleusercontent.com",  
  "sub": "110974621691774916798",  
  "email": "ct3ia77@gmail.com",  
  "email_verified": true,  
  "at_hash": "q0I_M5FDZGvgPF--0_RhHA",  
  "iat": 1769182562,  
  "exp": 1769186162  
}
```

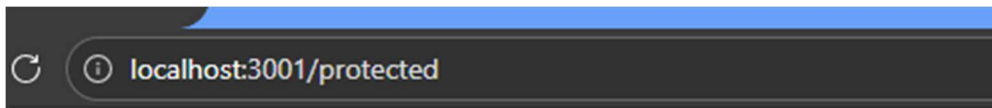
### 4.3 Cookie assinado (HMAC) e tentativa de adulteração

O cookie de sessão é assinado com HMAC usando SECRET\_KEY. Ao alterar o valor do cookie no browser ou ao mudar a chave, a assinatura deixa de validar e a aplicação deve rejeitar a sessão (forçando novo login).

[illegible]

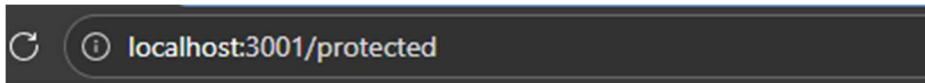
Com HMAC modificado





Authenticated as **ct3ia77@gmail.com** (signed cookie is valid)

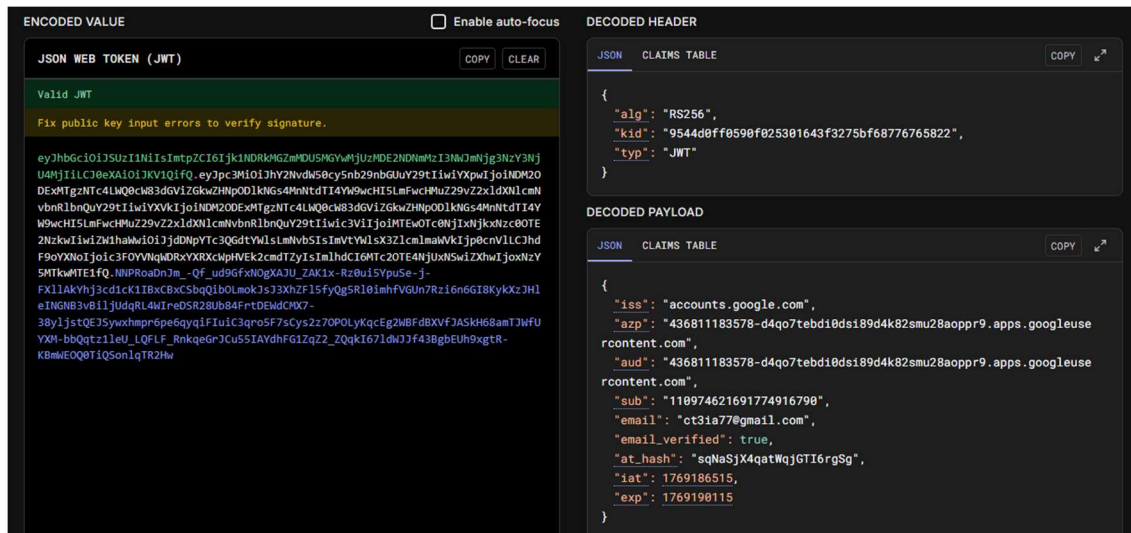
Com modificação da cookie no browser



Unauthorized: no valid signed cookie. Please login again.

## 4.4 Acesso à API Google Tasks (scope tasks.readonly)

O pedido inicial foi atualizado para incluir o scope de leitura de tasks (<https://www.googleapis.com/auth/tasks.readonly>). Na rota /callback foi adicionado um FORM para inserir o id de uma task list; a rota /tasklist usa o access\_token para chamar a API e listar títulos das tarefas.



### Limite de usuários do OAuth

Enquanto o status de publicação mostrar a opção "Testando", apenas os usuários de teste conseguirão acessar o aplicativo. O limite de usuários permitidos antes da verificação do app é de 100, contabilizado durante todo o ciclo de vida do app. [Saiba mais](#)

1 usuário (1 de teste, 0 outros) / limite de 100 usuários



### Usuários de teste

+ Add users

Filtro Insira o nome ou o valor da propriedade



#### Informações do usuário

ct3ia77@gmail.com



Ao adicionar o scope da Google Tasks API, foi necessário configurar o OAuth Consent Screen e adicionar o utilizador como "Test User", caso contrário o Google bloqueia o acesso com 403 access\_denied por a aplicação não estar verificada/publicada. Após esta configuração, o fluxo OAuth2/OIDC completou com sucesso.

[illegible]

Insert the **tasklistId** (get it from OAuth2 Playground or from /tasklists endpoint).

[List my tasklists \(helper\)](#)  
[Home](#)

- **Lista de Miguel** — MTMwOTY3NjkwOTczNjY0ODMyNTU6MDow

Tasks from list MTMwOTY3NjkwOTczNjY0ODMyNTU6MDow

- [Back](#)

O endpoint `/callback` foi modificado para, após a autenticação, apresentar um formulário para introdução do `tasklistId`. O formulário envia um POST para `/tasklist`, onde o `access_token` previamente obtido é utilizado para invocar a Google Tasks API (`tasks.list`) e devolver uma página HTML com os títulos das tarefas da lista selecionada.



## 4.6 Vulnerabilidade por ausência do parâmetro state

A ausência do parâmetro state torna o fluxo OAuth2/OIDC vulnerável a ataques de CSRF e authorization code injection (session swapping), permitindo que um atacante injete um authorization code no callback e associe indevidamente a sessão da vítima às credenciais do atacante.

1. O atacante autentica-se na Google e obtém um code válido.
2. O atacante envia à vítima um link:

`http://localhost:3001/callback?code=<code_do_atacante>`

3. A vítima abre o link (CSRF)
4. A app troca o code por tokens e cria sessão **como se fosse a vítima**, mas na verdade fica autenticada **na conta do atacante**.

## 5. Conclusões

Os laboratórios 1 e 2 permitiram evoluir uma API REST simples para uma solução com persistência relacional, autenticação segura (PBKDF2 + JWT), HTTPS e autorização RBAC. O trabalho prepara o projeto para a fase de testes (Lab3) e integração com clientes externos.

## Referências

- José Simão. Engenharia de Software Seguro - Laboratório 1 (Março 2024).
- José Simão. Engenharia de Software Seguro - Laboratório 2 (Janeiro 2026).
- Javalin SSL Tutorial (documentação oficial).
- Auth0 java-jwt (repositório oficial).
- Casbin jCasbin (repositório oficial).
- Google Tasks API (documentação Google).