

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)

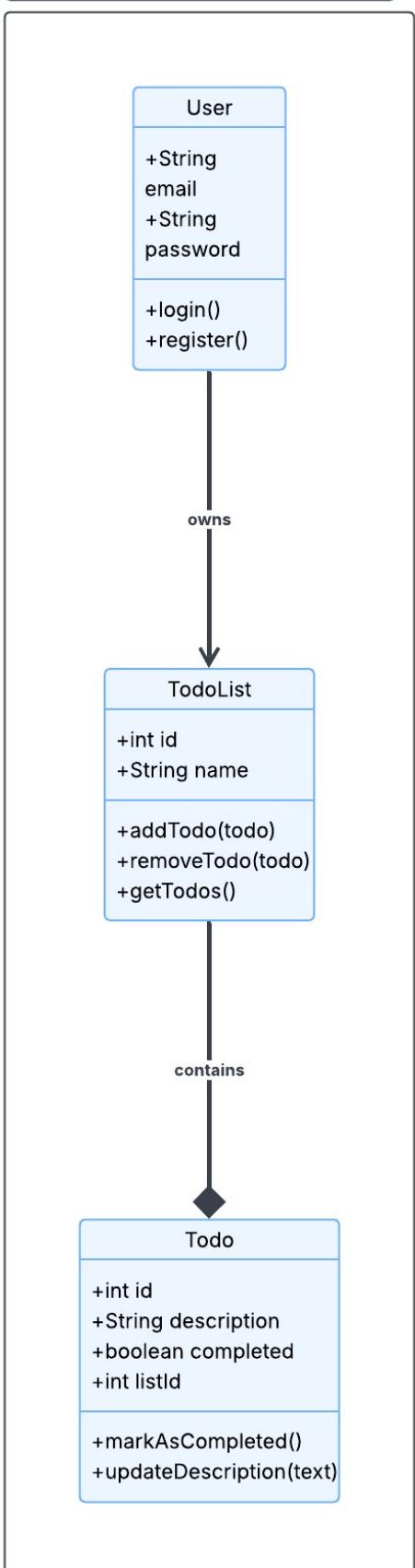
Como utilizador autenticado, quero que o sistema **verifique as minhas permissões** sempre que tento criar, ver, editar ou remover uma tarefa, para garantir que apenas eu ou utilizadores autorizados possamos manipular os meus dados.

- **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:

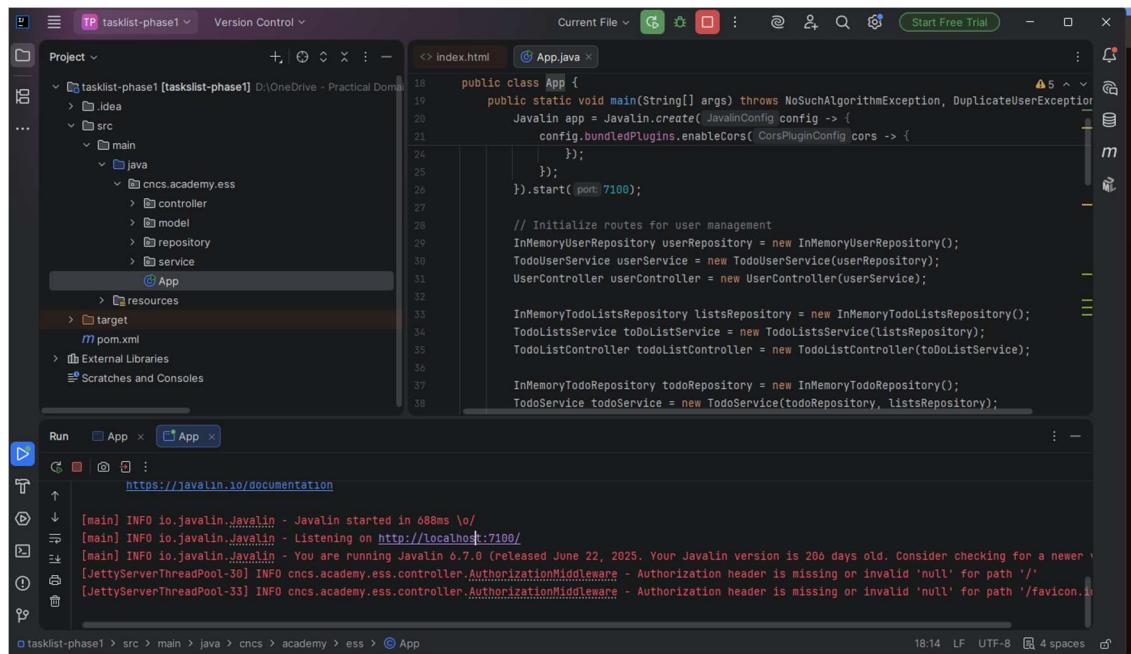
Todo App System



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

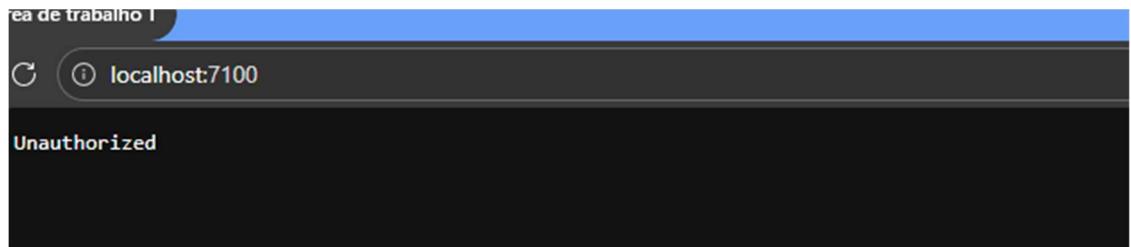
A aplicação foi executada localmente na porta 7100 e testada via Postman. O fluxo típico de testes foi:

- 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)



The screenshot shows a Java IDE interface with the following details:

- Project View:** Shows the project structure: tasklist-phase1 [tasklist-phase1] containing src, target, pom.xml, and External Libraries.
- Code Editor:** The file App.java is open, showing Java code for a Javalin application. It includes imports for Javalin, JavalinConfig, InMemoryUserRepository, TodoUserService, UserController, InMemoryTodoListsRepository, TodoListsService, TodoListController, InMemoryTodoRepository, and TodoService. The code sets up routes for user management, todo lists, and todos.
- Terminal:** Shows the command-line output of the application starting. It includes logs from Javalin and Jetty, indicating the app started at port 7100 and is listening on http://localhost:7100.
- Status Bar:** Shows the current time as 18:14, line endings as LF, encoding as UTF-8, and a 4 spaces indentation setting.



base1

HTTP ... / 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 }
```

POST Log • DEL Delete

No environment

HTTP ... / Login user Save Share | [Link](#)

POST localhost:7100/login Send

Body Cookies

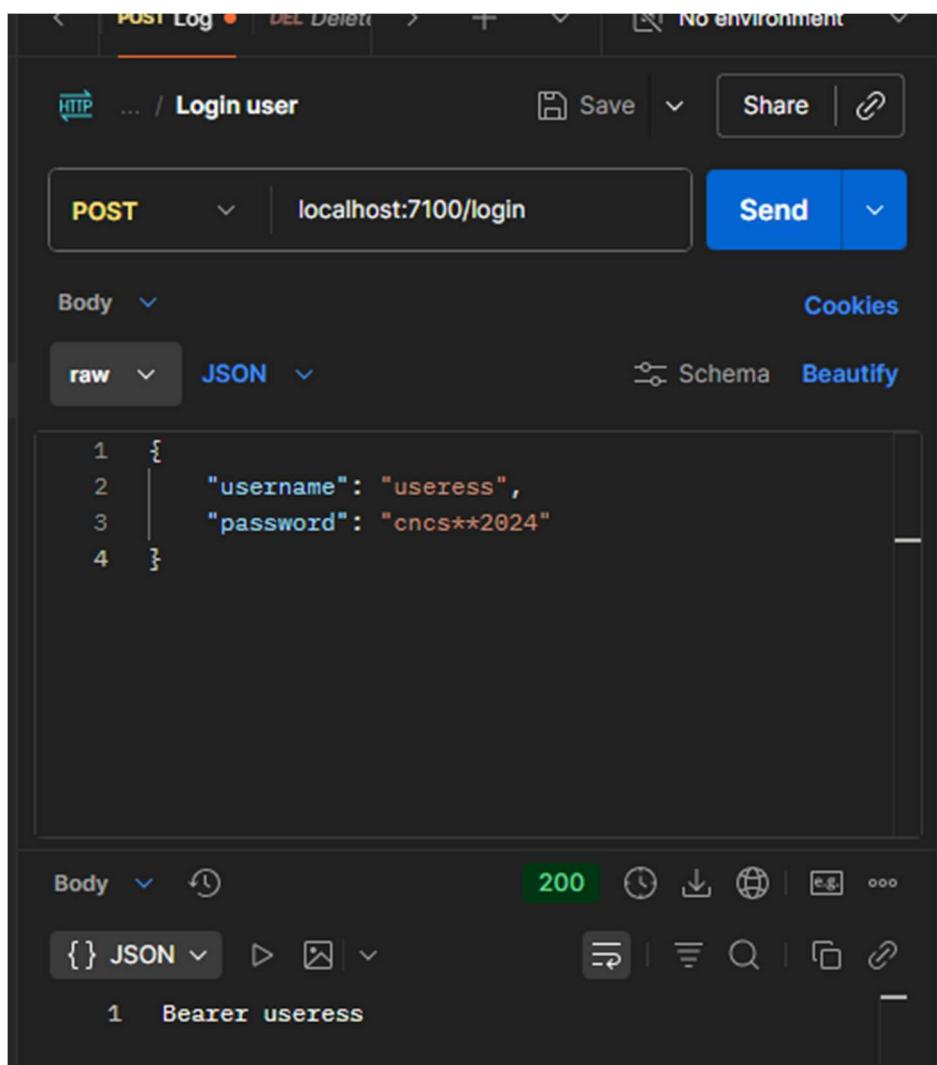
raw JSON Schema Beautify

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

Body 200 [Clock](#) [Download](#) [World](#) [Copy](#) [More](#)

{ } JSON [Preview](#) [Visualize](#) [Copy](#) [Link](#)

1 Bearer useress



HTTP C-Academy ESS - Todo Lists - Phase1 / Lists / Get All Lists of user Save Share | [Link](#)

GET localhost:7100/todolist Send

Docs Params Authorization Headers (8) Body Scripts Settings Cookies

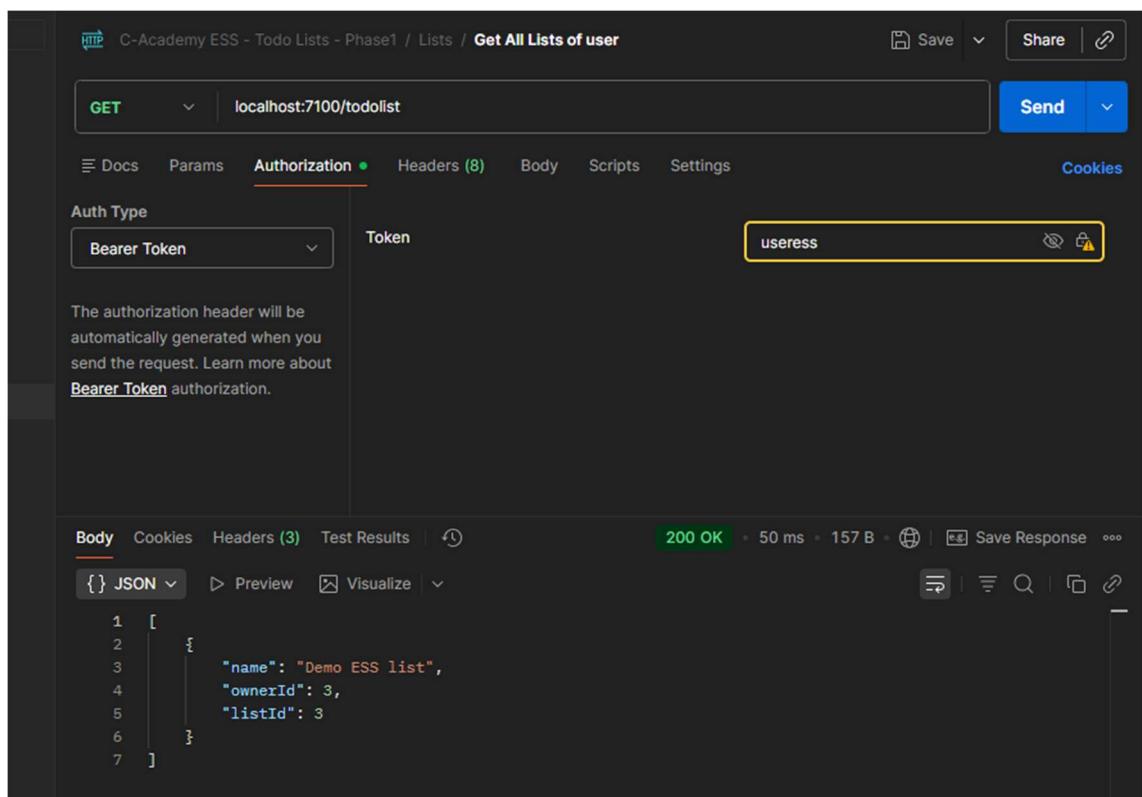
Auth Type: Bearer Token Token: useress

The authorization header will be automatically generated when you send the request. Learn more about [Bearer Token](#) authorization.

Body Cookies Headers (3) Test Results [Copy](#) 200 OK 50 ms 157 B [Save Response](#) [More](#)

{ } JSON [Preview](#) [Visualize](#) [Copy](#) [Link](#)

```
1 [  
2   {  
3     "name": "Demo ESS list",  
4     "ownerId": 3,  
5     "listId": 3  
6   }  
7 ]
```



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.

The screenshot shows the Postman application interface. At the top, there's a toolbar with various icons and a dropdown menu. Below it, the main header bar displays "POST Create" and the URL "localhost:7100/user". On the right side of the header, there are "Save", "Share", and a copy icon. The main content area is divided into sections: "Body" (with "raw" and "JSON" tabs), "Cookies", and "Schema/Beautify". The "Body" tab is active, showing the following JSON payload:

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

Below this, there's a large text area for the response body. At the bottom, there are "Body" and "JSON" dropdowns, and a status bar showing "201". The status bar also includes icons for refresh, download, globe, and more.

The response body is displayed in a separate panel at the bottom, showing the created user object:

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

	id [PK] integer	username character varying (255)	password character varying (255)
1	1	user1	password1
2	2	user2	password2
3	3	useress	cncts**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 accidental 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
main] INFO io.javalin.Javalin - Exception in thread "main" java.lang.RuntimeException Create breakpoint : Failed to save user
at cncts.academy.ess.repository.sql.SQLUserRepository.save(SQLUserRepository.java:66)
```

The screenshot shows a Postman interface with the following details:

- Method:** POST
- URL:** <https://localhost/user>
- Body (JSON):**

```
1  {
2    "username": "useres51",
3    "password": "cncts**2024"
4 }
```
- Response Status:** 201 Created
- Response Body (JSON):**

```
1  {
2    "userId": 21,
3    "username": "useres51"
4 }
```

The authorization header will be automatically generated when you send the request. Learn more about [Bearer Token](#) authorization.

```

1  {
2    "listId": 1,
3    "name": "Demo ESS list",
4    "userId": 6
5  }

```

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();

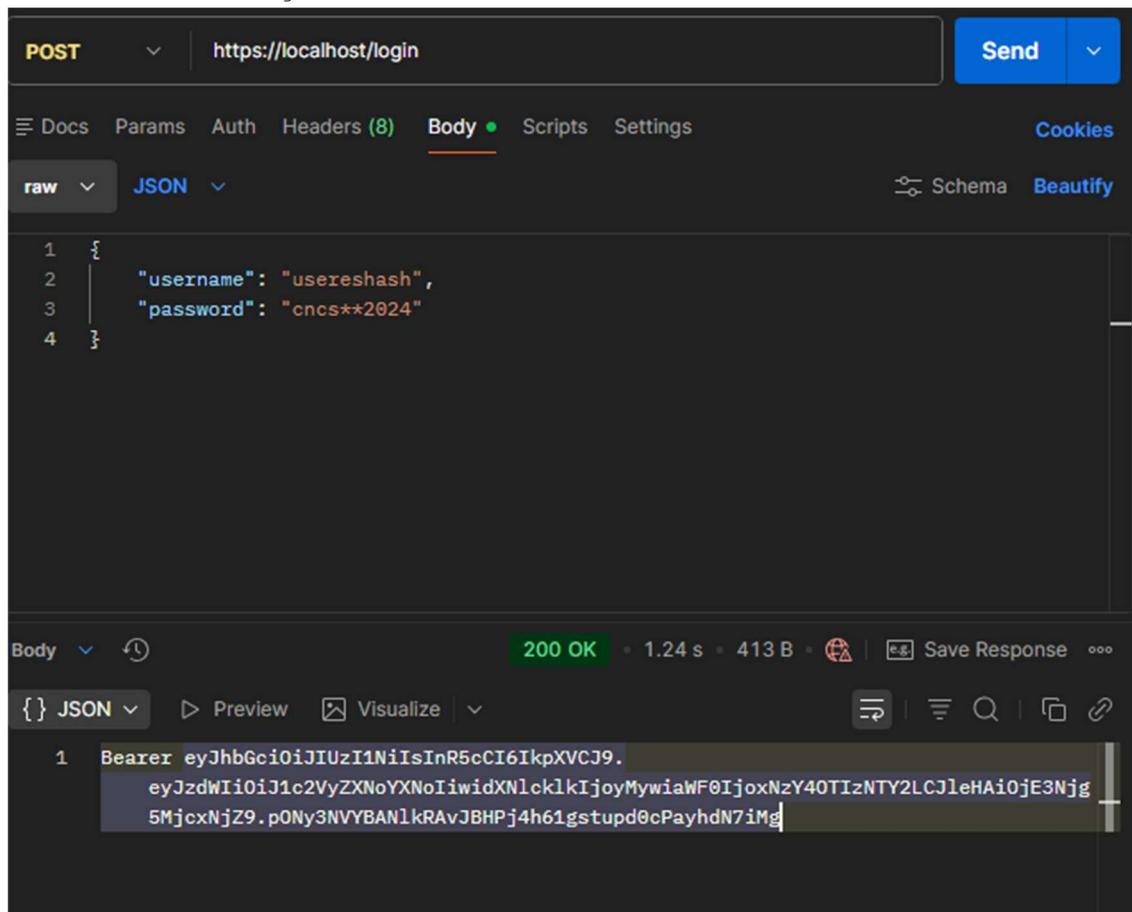
```

Data Output				Messages	Notifications
	<input data-bbox="203 1567 250 1599" type="button" value="+"/> <input data-bbox="257 1567 289 1599" type="button" value="F"/> <input data-bbox="295 1567 327 1599" type="button" value="V"/> <input data-bbox="333 1567 365 1599" type="button" value="C"/> <input data-bbox="371 1567 403 1599" type="button" value="D"/> <input data-bbox="409 1567 441 1599" type="button" value="S"/> <input data-bbox="447 1567 479 1599" type="button" value="A"/>	Showing rows: 1 to 6			
	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	
1	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
2	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
3	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
4	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
5	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
6	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

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;
```



The screenshot shows a Postman interface with the following details:

- Method:** POST
- URL:** <https://localhost/login>
- Headers:** (8 items)
- Body:** (JSON)
```json  
1 {  
2 "username": "usereshash",  
3 "password": "cncs\*\*2024"  
4 }  
```
- Response Status:** 200 OK
- Response Time:** 1.24 s
- Response Size:** 413 B
- Response Content:** Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiJ1c2VyZXNoYXNoIiwidXNlcklkIjoyMywiaWF0IjoxNzY4OTIzNTY2LCJleHAiOjE3Njg5MjcxNjZ9.pONy3NVYBANlkRAvJBHPj4h61gstupd0cPayhdN7iMg|

The screenshot shows a Postman interface with the following details:

- Method:** GET
- URL:** https://localhost/todolist
- Headers:** Authorization (Value: Bearer eyJhbGciOiJIUzI1NiIsInR5c...)
- Body:** JSON response (200 OK):


```

1 [ 
2   {
3     "id": 3,
4     "name": "Demo ESS list user com hash",
5     "ownerId": 23,
6     "listId": 3
7   }
8 ]
      
```
- Response Headers:** 200 OK, 166 ms, 307 B, Save Response

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, todolist, create
p, role_base, todolist, read

p, role_base, todoitem, create
p, role_base, todoitem, read
p, role_base, todoitem, delete

# -----
# SHARE role: Base + share lists
# -----
p, role_share, todolist, 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 pode 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 0.1.4 (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:
```

The screenshot shows the Postman application interface. At the top, a header bar displays the URL `https://localhost/user`. Below this, the main interface has several tabs: Docs, Params, Auth, Headers (8), Body (selected), Scripts, Settings, and Cookies. The Body tab is currently active and set to JSON format. It contains the following JSON payload:

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

On the right side of the Body panel, there are buttons for Schema and Beautify. Below the Body panel, the Response section is visible, showing a status of 201 Created, a response time of 229 ms, and a response size of 273 B. The response body is displayed as:

```
1 {  
2   "userId": 30,  
3   "username": "admin"  
4 }
```

Below the response, there are various icons for managing the request and response.

user com role_base → 403

user com role_share → 200

admin → 200

4. Laboratório 2 - OAuth2/OIDC com Google Tasks

4.1 Configuração do cliente na Google Cloud Console

Foi desenvolvida uma aplicação web Node.js que autentica utilizadores com o Google usando OpenID Connect e mantém a sessão via cookies assinados com HMAC (SECRET_KEY).

Após autenticação, foi analisado o id_token (JWT) e validado o conteúdo do payload (incluindo email).

O scope foi alargado para incluir <https://www.googleapis.com/auth/tasks.readonly> e a aplicação passou a consumir a API Google Tasks, obtendo tarefas de uma lista indicada pelo utilizador via formulário na rota /callback e apresentando-as em /tasklist.

Por fim, identificou-se que a ausência do parâmetro state torna o fluxo vulnerável a ataques do tipo CSRF/authorization code injection (session swapping).

4.2 Análise do id_token (JWT)

Após autenticação, a aplicação imprime code, access_token e id_token. O id_token é um JWT que pode ser analisado (header/payload/signature) para observar claims como iss, aud, exp e email.

callback with code:

= 4/0ASc3gC1uO79psj8tciqGumvKZd3pEHNcjXsSRfLGY29VfowQJFHuR35q2QEIAr_Qdjx4JQ

client app received access code:

= ya29.a0AUMWg_K74b6VHkBan4mDgvhGmMrEg1TLYa8KPVtfOTEm5Xo_OJZxhNhEWcVHW3yOnAx54N023QN0tV41_ous6kiMjJDLxOfI9liKZLlw0wIR2QWiFXBNkFLUxWiMeTEfSCu29EFRImSCsRZe7voio_k_ckwaRcgcs27z8klrEfgeOMXxKPAb1GTdQiQhGNdERg2fCEaCgYKAUkSARUSFQHGX2MiWRPICZVZ-II8_EGPTQcX-g0206

id_token

= eyJhbGciOiJSUzI1NiIsImtpZCI6Ijk1NDRkMGZmMDU5MGYwMjUzMDE2NDNmMzI3NWJmNjg3NzY3NjU4MjliLCJ0eXAiOiJKV1QifQ.eyJpc3MiOiJhY2NvdW50cy5nb29nbGUuY29tliwiYXpwIjoiNDM2ODExMTgzNTc4LWQ0cW83dGVizGkwZHNpODIkNGs4MnNtdTI4YW9wcHI5LmFwcHMuZ29vZ2xIdXNlcmNvbnRlbnQuY29tliwiYXVkljoiNDM2ODExMTgzNTc4LWQ0cW83dGVizGkwZHNpODIkNGs4MnNtdTI4YW9wcHI5LmFwcHMuZ29vZ2xIdXNlcmNvbnRlbnQuY29tliwi3ViljoiMTEwOTc0NjlxNjkxNzc0OTE2NzkwljwiZW1haWwiOijjdDNpYTc3QGdtYWlsLmNvbSIslmVtYWlsX3ZlcmlmaWVkljp0cnVILCJhdF9oYXNoljoicTBJX001RkRaR3ZnUEYtMC1fUkhoQSISlmlhdCI6MTc2OTE4MjU2MiwiZXhwIjoxNzY5MTg2MTYyfQ.Yd1lsIZ5OLKJXcPe75DsxAzNkGGkl-9J7rSFvMgKla4B9RSxVSWTw0zxsuRjU8ghzL6JbNhbcB5FOAsuiWM9OtVxWGITW1sI-juPpLxtRTUIw0Ip8s-9Z0aB2Z8alx4y5Vx_1IY0_68wgNKSsGubfq7roQYHwdSGSTrTn27-G7fIJToA-Z4pBU4tg8_8L9kpVPwDUOTrz4GoH9ePNlp1PhsOykZaShC_Iz94GzFl-7gdCveloJNBt-MFutZ-75SKYS2WoISbXy6qXjXmCCmQOzWudiZQG8htdZ1KLTd7nSFj30sQEYK5OIZEAQZugLFjYaAzoexUq5_SgtQMIsuiQww

Hi ct3ia77@gmail.com

Go back to [Home screen](#)

ENCODED VALUE	DECODED HEADER	
<input checked="" type="checkbox"/> Enable auto-focus JSON WEB TOKEN (JWT) Valid JWT Fix public key input errors to verify signature. eyJhbGciOiJSUzIiNiImsMpZC16Ijk1NDRkMGZmMDUSMgywMjUzMDE2NDNmMzI3NvJgnNg3NzY3NjU4Mj1lC0eXA10IjV1QfQ_fyJp3M10IjHjY2Nvd58cy5nb29nbGUvY29tIiwiXpwjjojnDM20DEXMtgzNTc4LWQ8cW83dGV1ZGwzHnp001KNGs4MnNtdT14YW9wcH5LmfwcHMuZ29vZ2xlDXnlcmNbmrR1bnQuV29tIiwiYXK1joINM20DEMTgNTc4LWQ8cW83dGV1ZGwzHnp001KNGs4MnNtdT14YK9uCHISLmFwCHMuZ9vZx1dXN1cmNbmrR1bnQuV29tIiwiYijoIMTewOTc0NjIxkjxzc00TE2NzkwIiwiZM1haw101jjdWPyTc3QgdTYmlsLmVbS1sImVtY1sX321cm1mawVkjIj0cVn1lC3hdF9oYXNoIjoiCTBX001RkrAr32JUEYMC1fUkhoQS1sIm1hdC16MtC20TE4Mju2MiwiZkhw1joxNz5SMtgMNYfQ_fyJ11st750LK1KcPe75dsxA2Nkgk1-937r5FvMgKia489RSxVSWTw0zsuhj08ghzL6jbNhbcB5FOAsu1wM90tVxwGITW1sI-JubPplxtRTUin0tp8s-920a82z8aIXdy5Vx_1IV_68wgNKGubfqr7rQHwdSGSTrTn27-G7f1f0A-Z4pB04tg8_8L9kpVpwDU0trz4GOh9ePNlp1PhsOykzaShC_1z94Gzf1-7gcCveloJNBT-MfutZ-75SKYS2Wo1sby6oXjXmcQ0zNudi2Q68htd21kL1d7nSFj30sQEyK50I2EAQZugLFjYaAzoexUq5_SgtQMlsuiQvw	<input checked="" type="checkbox"/> COPY <input checked="" type="checkbox"/> CLEAR JSON CLAIMS TABLE { "alg": "RS256", "kid": "9544d8ff0590f025301643f3275bf68776765822", "typ": "JWT" } DECODED PAYLOAD <td><input checked="" type="checkbox"/> COPY <input checked="" type="checkbox"/> CLEAR JSON CLAIMS TABLE { "iss": "accounts.google.com", "azp": "436811183578-d4qo7tebdio189d4k82smu28aoppr9.apps.googleusercontent.com", "aud": "436811183578-d4qo7tebdio189d4k82smu28aoppr9.apps.googleusercontent.com", "sub": "118974621691774916798", "email": "ct3ia77@gmail.com", "email_verified": true, "at_hash": "q01_M5FDZGvgPF-0_-RHhA", "iat": 1769182562, "exp": 1769186162 }</td>	<input checked="" type="checkbox"/> COPY <input checked="" type="checkbox"/> CLEAR JSON CLAIMS TABLE { "iss": "accounts.google.com", "azp": "436811183578-d4qo7tebdio189d4k82smu28aoppr9.apps.googleusercontent.com", "aud": "436811183578-d4qo7tebdio189d4k82smu28aoppr9.apps.googleusercontent.com", "sub": "118974621691774916798", "email": "ct3ia77@gmail.com", "email_verified": true, "at_hash": "q01_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).

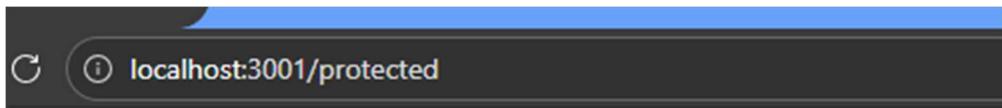
Name	Value	Dom...	Pat...
continueCode	bKZ8yYJg36bORLxl5zMNanDAoVtbfy...	local...	/
cookieconsent_status	dismiss	local...	/
DemoCookie	s%3Act3ia77%40gmail.com.94klv9qfs...	local...	/
language	en	local...	/
welcomebanner_status	dismiss	local...	/

Cookie Value Show URL-decoded
sct3ia77@gmail.com.94klv9qfshotukLxaJGy3QChwA7zzWEM+sFppsNf+zQ

Com HMAC modificado

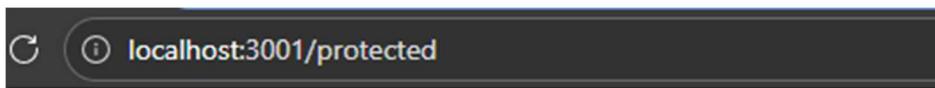


Unauthorized: no valid signed cookie. Please login again.



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.

The screenshot shows a JWT (JSON Web Token) being decoded. The 'Encoded Value' section contains a long string of base64-encoded data. The 'Decoded Header' section shows the following JSON:

```
{
  "alg": "RS256",
  "kid": "9544d0ff0590f025301643f3275bf68776765822",
  "typ": "JWT"
}
```

The 'Decoded Payload' section shows the following JSON:

```
{
  "iss": "accounts.google.com",
  "azp": "436811183578-d4qo7tebd10ds189d4k82smu28aoppr9.apps.googleusercontent.com",
  "aud": "436811183578-d4qo7tebd10ds189d4k82smu28aoppr9.apps.googleusercontent.com",
  "sub": "110974621691774916798",
  "email": "ct3ia77@gmail.com",
  "email_verified": true,
  "at_hash": "sqNaSjX4qatWqjGTI6rgSg",
  "iat": 1769186515,
  "exp": 1769190115
}
```

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.

4.5. Acesso a listas

```
callback with code = 4/0ASc3gC0aGws-4hT6ixXwFvn61w28yvPJc6mSsR7fi4gi_xNzbDHQUrUuQobNGr1_IZqSA
access_token = ya29.a0AUWNg_LUSNK3c-OuHn-Sse_QPCK_w93GVQTs0c0QN9_Ty50vIyrIM7VGt82TNNTVBakLgYedGvvLuNrab071I23Iz6PvX7jPwxBuM0VY01xRqT-
THK12on7M_KwxUhLUKFphQDYLoyUek7RxJluSom2q50fZyaxp2kkRXZ1IAIKSP7gwI5X8tZ9ldg1DPFv5mhA-YaCgYKaBESARUFQHQX2HihejQ_sofhXzvZyhCBBRuA0206
id_token =
eyJhbGciOiJSUzI1NiIsImtpZCI6Ijk1NDRkMGZmNDU5MGYwMjUzMDE2NDNmMzI3NwJmNjg3NzY3NjU4MjIiLCJ0eXAiOiJKV1QiJQ.eypc3Mi0iJhY2Nvdw50cy5nb29nbGUuY29tIiwiYXpwIjoiNDH20DExMtgzNTc4LwQ0cWB3dGV1
xxr0xY58btMmTQjRPcGe5QyvAm79_nu1BVbcwJPF4Hew-Ga5yoGmXB9hL6xhfkx_04HR3PPmit8HiByAOVJ-dwIqlf5guadZVvI2UdUucvBymM0x4CABD1cI7J2rYUOZ#Hea-
jctig4d02CRNRb07pxi6vx1HY31jPr057U9FJVGhV5cHYZvBK3A1z6u6nrVw4jI5iybTv33n1VHmTmXGsFvM-0KnchE71cQLkJME6-Z6wnJvs54v1zR1dmmtq2MS2c-l_t6n_CHzi1xN7VGzShLRNKRucv02-ccgL7z5rd7w81w2u6dH
Hi ct3ia77@gmail.com
```

Choose a Tasklist

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

Tasklist ID:

[Get tasks](#)

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

Your Tasklists

- **Lista de Miguel** — MTMwOTY3NjkOTczNjY0ODMyNTU6MDow

[Back](#)

Tasks from list MTMwOTY3NjkOTczNjY0ODMyNTU6MDow

- Verificar TVs que dá para consertar
- Enviar material para o Lixo
- Desarmar TVs na arrecadação
-
- Falta pouco para o seu anúncio ser publicado!
-

[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).

Engenharia de Software Seguro

Relatório - Laboratório 3

Testes (unitários e integração) e aplicação cliente multiplataforma

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): lab3

Data: 06-02-2026

Sumário

1. Testes unitários (JUnit + Mockito) e cobertura
2. Testes de integração (Postman + Newman via Docker)
3. Aplicação cliente Flutter
4. Checkpoint do projeto

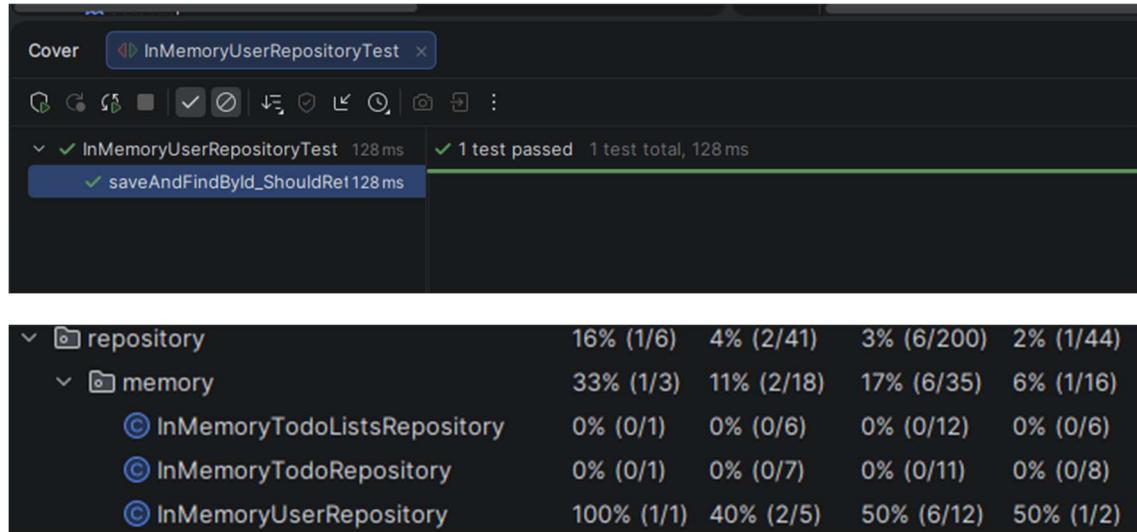
1. Testes unitários (JUnit + Mockito) e cobertura

1.1 Dependências e estrutura do projeto

O projeto Maven inclui dependências de teste para JUnit Jupiter e Mockito. Os testes devem ficar em `src/test/java`, seguindo a convenção Maven.

1.2 Teste base: InMemoryUserRepositoryTest e cobertura

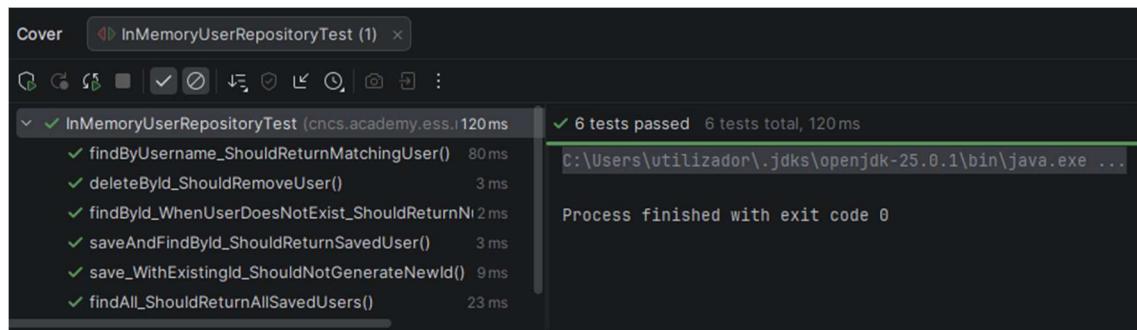
Foi executado o teste fornecido para o repositório em memória (`saveAndFindById_ShouldReturnSavedUser`) com medição de coverage.



1.3 Testes adicionais para aumentar a cobertura

Foram acrescentados testes para cobrir cenários adicionais e aumentar a cobertura do repositório em memória, por exemplo:

- `findByUsername_ShouldReturnUserWhenExists`
- `findByUsername_ShouldReturnNullWhenMissing`
- `deleteById_ShouldRemoveUser`
- `save_ShouldAssignIncrementalId`
- [outros testes relevantes]



Element		Class, %	Method, %	Line, %	Branch, %
cncts.academy.ess.repository.memory	33% (1/3)	27% (5/18)	34% (12/35)	12% (2/16)	
↳ InMemoryTodoListsRepository	0% (0/1)	0% (0/6)	0% (0/12)	0% (0/6)	
↳ InMemoryTodoRepository	0% (0/1)	0% (0/7)	0% (0/11)	0% (0/8)	
↳ InMemoryUserRepository	100% (1/1)	100% (5/5)	100% (12/12)	100% (2/2)	

1.4 Porque usar mocks no TodoUserService e não no InMemoryUserRepository

No teste do InMemoryUserRepository pretende-se validar o comportamento real de persistência em memória (estado interno, geração de IDs, pesquisas), pelo que não faz sentido mockar a própria classe sob teste.

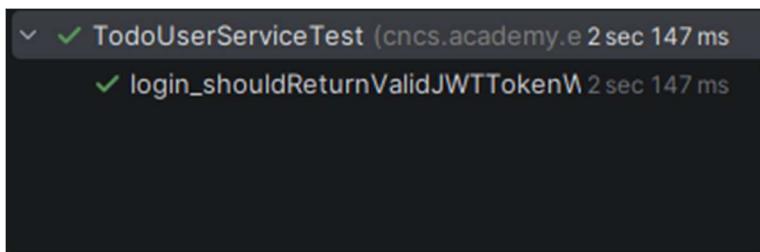
Já no TodoUserService, o objetivo é testar regras de negócio (ex: login, criação de utilizadores) de forma isolada e determinística, sem depender do repositório concreto, da base de dados, ou de detalhes externos. Mocks permitem simular respostas do UserRepository, forçar erros e validar interações.

1.5 Teste com Mockito ao método login e validação do JWT

Foi implementado o teste login_shouldReturnValidJWTTokenWhenCredentialsMatch. O teste prepara um UserRepository mock para devolver um utilizador existente e invoca login com credenciais corretas. Depois valida:

- (1) o retorno começa por 'Bearer'
- (2) o restante é um JWT válido, com assinatura e claims esperadas (issuer, username, issuedAt, expiresAt).

```
org.opentest4j.AssertionFailedError: claim 'username' deve corresponder ao utilizador autenticado ==>
Expected :userress1
Actual   :userress5
<click to see difference>
```



2. Testes de integração (Postman + Newman via Docker)

2.1 Execução de coleções Postman com Newman

As coleções Postman dos laboratórios anteriores foram adaptadas para execução em linha de comandos usando Newman, via imagem Docker postman/newman. Foi necessário substituir 'localhost' por 'host.docker.internal' e mapear o diretório da coleção para dentro do contentor.

```
# Executar a coleção (no diretório onde está o ficheiro .json)
docker run -v ${PWD}:/etc/newman -t postman/newman run newman-v1.2.json
```

C-Academy ESS - Todo Lists

```
▢ Users
↳ Create User
  POST host.docker.internal:80/user [500 Server Error, 252B, 188ms]
    1. User created or exists

↳ Login user
  POST host.docker.internal:80/login [200 OK, 411B, 89ms]
    ✓ Login successful

→ Create a new list
  POST host.docker.internal:80/todolist [201 Created, 297B, 21ms]
    ✓ List created successfully

→ Get All Lists
  GET host.docker.internal:80/todolist [200 OK, 368B, 17ms]
    ✓ Status code is 200
    ✓ Response is an array
```

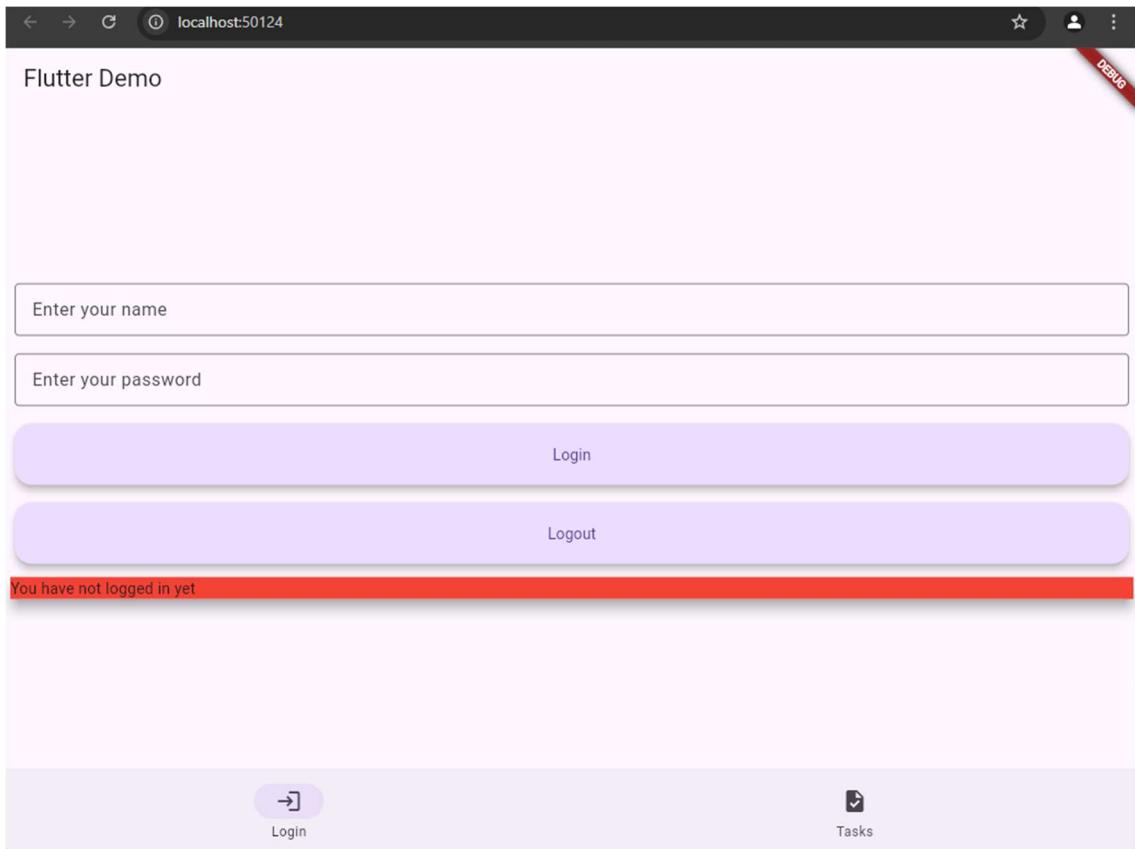
	executed	failed
iterations	1	0
requests	4	0
test-scripts	4	0
prerequest-scripts	0	0
assertions	5	1
total run duration: 450ms		
total data received: 373B (approx)		
average response time: 78ms [min: 17ms, max: 188ms, s.d.: 69ms]		

```
# failure           detail
1. AssertionError      User created or exists
                           expected 500 to be one of [ 201, 400, 409 ]
                           at assertion:0 in test-script
                           inside "Users / Create User"
```

3. Aplicação cliente Flutter

3.1 Compilação e execução em web

O ambiente Flutter foi instalado e a aplicação base (todoapp) foi compilada e executada em modo web. Foi registado o ecrã inicial de login.



3.2 Implementação de todo_service.dart

Foram completados os métodos da classe TodoService para chamar a API:

- Usar a variável de ambiente HOST para compor o endereço base
- Reutilizar o token guardado após login no header Authorization
- Mapear respostas JSON para os modelos TodoListModel e TodoModel

```
INFO cncts.academy.ess.controller.UserController - Login user: useress5
INFO org.casbin.jcasbin - Request: user, todolist, read ---> false
INFO cncts.academy.ess.controller.TodoListController - Get all todo lists

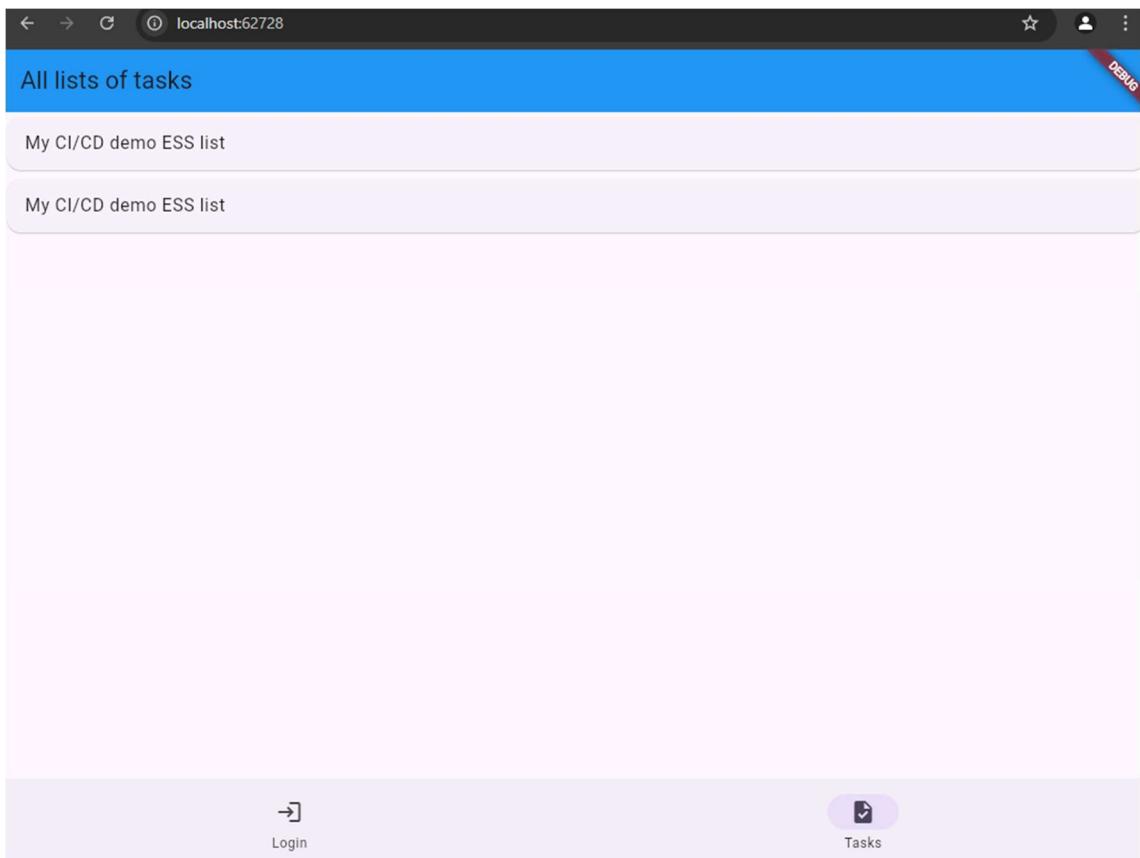
This app is linked to the debug service: ws://127.0.0.1:49693/TlIf-LATrAU=/ws
Debug service listening on ws://127.0.0.1:49693/TlIf-LATrAU=/ws
A Dart VM Service on Chrome is available at: http://127.0.0.1:49693/TlIf-LATrAU=
The Flutter DevTools debugger and profiler on Chrome is available at:
http://127.0.0.1:49693/TlIf-LATrAU=/devtools/?uri=ws://127.0.0.1:49693/TlIf-LATrAU=/ws
Starting application from main method in: org-dartlang-app:/web_entrypoint.dart.

#0 package:todoapp/services/logger.dart 16:48                      info
#1 package:todoapp/screens/login_screen.dart 116:19                  <fn>

💡 Username: useress5, Password: cncts**2026
```

```
| POST http://localhost/login -> 200
|
|
| #0 package:todoapp/services/logger.dart 16:48 info
| #1 package:todoapp/services/todo_service.dart 32:21 <fn>
|
|💡 Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJ1c2VyZXNzNSIsInVzZXJJZCI6MzgsImlhdcI6MTc2OTM2MzU0MCwiZXhwIjoxNzY5MzY3MTQwfQ.n5vwLC-u5cDevKjtTMEkvvYWaBj2_wZiSqA3vsXAzvs
|
|
| #0 package:todoapp/services/logger.dart 16:48 info
| #1 package:todoapp/screens/login_screen.dart 118:19 <fn>
|
|💡 Result: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJ1c2VyZXNzNSIsInVzZXJJZCI6MzgsImlhdcI6MTc2OTM2MzU0MCwiZXhwIjoxNzY5MzY3MTQwfQ.n5vwLC-u5cDevKjtTMEkvvYWaBj2_wZiSqA3vsXAzvs
```

```
| #0 package:todoapp/services/logger.dart 16:48 info
| #1 package:todoapp/services/todo_service.dart 51:19 <fn>
|
|💡 GET http://localhost/todolist with token <Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJ1c2VyZXNzNSIsInVzZXJJZCI6MzgsImlhdcI6MTc2OTM2MzU0MCwiZXhwIjoxNzY5MzY3MTQwfQ.n5vwLC-u5cDevKjtTMEkvvYWaBj2_wZiSqA3vsXAzvs>
|
|
| #0 package:todoapp/services/logger.dart 16:48 info
| #1 package:todoapp/services/todo_service.dart 61:21 <fn>
|
|💡 Response (200): [{"id":4,"name":"My CI/CD demo ESS list","ownerId":38,"listId":4}, {"id":5,"name":"My CI/CD demo ESS list","ownerId":38,"listId":5}]
```



3.3 Armazenamento seguro do token (`flutter_secure_storage`)

A biblioteca `flutter_secure_storage` pode ser usada para guardar o token num armazenamento seguro do sistema (Keychain no iOS, Keystore no Android). O token é gravado após login e lido antes de chamadas futuras, evitando que fique apenas em memória. Em web, a estratégia deve ser avaliada com cuidado (armazenamentos web são mais expostos).

Referências

- José Simão. Engenharia de Software Seguro - Laboratório 3 (Janeiro 2026).
- JUnit Jupiter (documentação oficial).
- Mockito (documentação oficial).
- Postman Newman (documentação oficial).
- Flutter (documentação oficial).