

PMR 3412 - Redes Industriais

# Criptografia na Internet

Gustavo Marangoni Rubo - 4584080

## 1 Exercício 1 - Senhas e Login

### 1.1 API RESTful de usuários

Foi criado um servidor em python que recebe requisições REST para gerenciar um banco de dados de usuários. As requisições foram feitas com a extensão "RESTED Client" do firefox. Segue o código do servidor:

```
1 from flask import Flask
2 from flask import request
3 import json
4
5 app = Flask(__name__)
6
7 # Inicializando a lista de usuários
8 users = [
9     {
10         "user_id": 1,
11         "user_name": "Gustavo Rubo",
12         "user_password": "4584080",
13         "user_email": "gustavo.rubo@usp.br"
14     }
15 ]
16
17 id_counter = 2
18
19 # Rota para coleção de usuários
20 @app.route("/users", methods=["POST", "GET", "PUT", "DELETE"])
21 def process_users():
22     global id_counter, users
23     if request.method == "POST":
24         # Criar um usuário e adicioná-lo à lista
25         user = {
26             "user_id": id_counter,
27             "user_name": request.json["user_name"],
28             "user_password": request.json["user_password"],
29             "user_email": request.json["user_email"]
30         }
31         id_counter += 1
32         users.append(user)
33
34         return json.dumps(user), 201
35
36     elif request.method == "GET":
37         # Retornar todos os usuários
38         return json.dumps(users), 200
39
40     elif request.method == "PUT":
41         return "Method not allowed", 405
42     elif request.method == "DELETE":
43         return "Method not allowed", 405
44
45 # Rota específica para usuários
46 @app.route("/users/<user_id>", methods=["POST", "GET", "PUT", "DELETE"])
47 def process_users_id(user_id):
48     global id_counter, users
49     if request.method == "POST":
50         return "Method not allowed", 405
51
52     elif request.method == "GET":
```

```

53     # Retornar o usu rio com o id especificado
54     user = next((u for u in users if u["user_id"] == int(user_id)), None)
55
56     if user:
57         return json.dumps(user), 200
58     else:
59         return "User not found", 404
60
61 elif request.method == "PUT":
62     # Modificar o usu rio com o id especificado
63
64     user = next((u for u in users if u["user_id"] == int(user_id)), None)
65
66     if user:
67         user["user_name"] = request.json["user_name"]
68         user["user_password"] = request.json["user_password"]
69         user["user_email"] = request.json["user_email"]
70         return json.dumps(user), 200
71     else:
72         return "User not found", 404
73
74 elif request.method == "DELETE":
75     # Remover o usu rio com o id especificado
76     user = next((u for u in users if u["user_id"] == int(user_id)), None)
77
78     if user:
79         users[:] = [u for u in users if u["user_id"] != int(user_id)]
80         return "User deleted succesfully", 200
81     else:
82         return "Usu rio n o encontrado", 404
83
84 # Rota para login de usu rio
85 @app.route("/login", methods=["POST"])
86 def process_login():
87     global users
88     if request.method == "POST":
89         user_name = request.json["user_name"]
90         user_password = request.json["user_password"]
91         user = next((u for u in users if u["user_name"] == user_name), None)
92
93         if user["user_password"] == user_password:
94             return json.dumps({"login": "true"}), 200
95         else:
96             return json.dumps({"login": "false"}), 200

```

Em seguida, foram feitas as requisições teste para validar o funcionamento do servidor. Todas as requisições mostradas através dos pintscreens a seguir foram feitas sucessivamente.

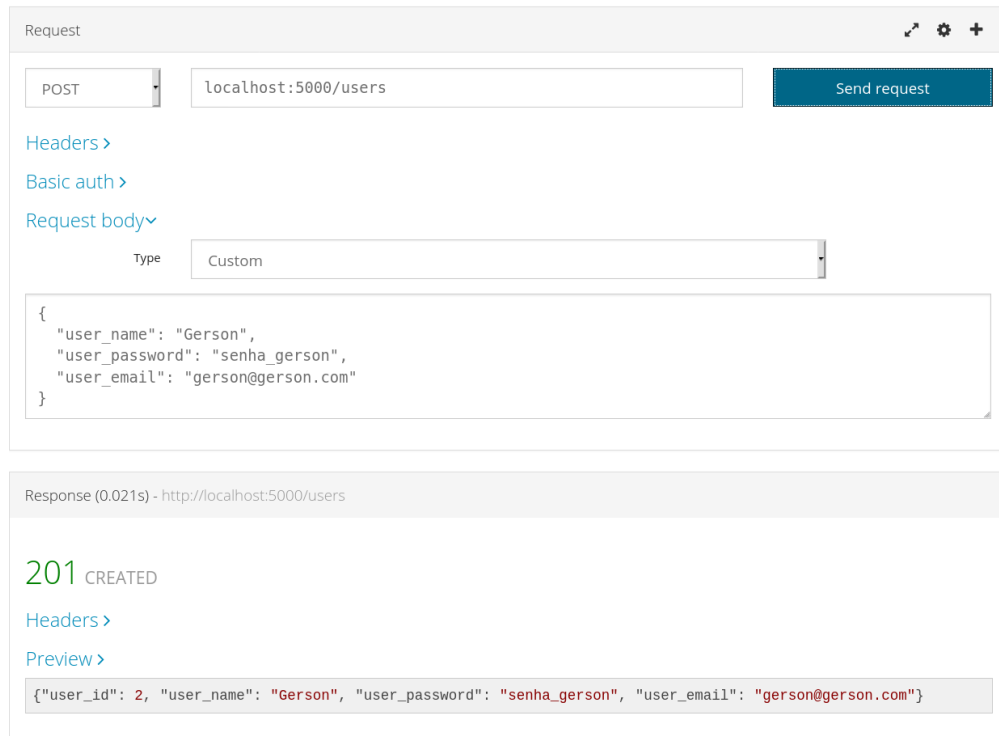


Figura 1: POST /users

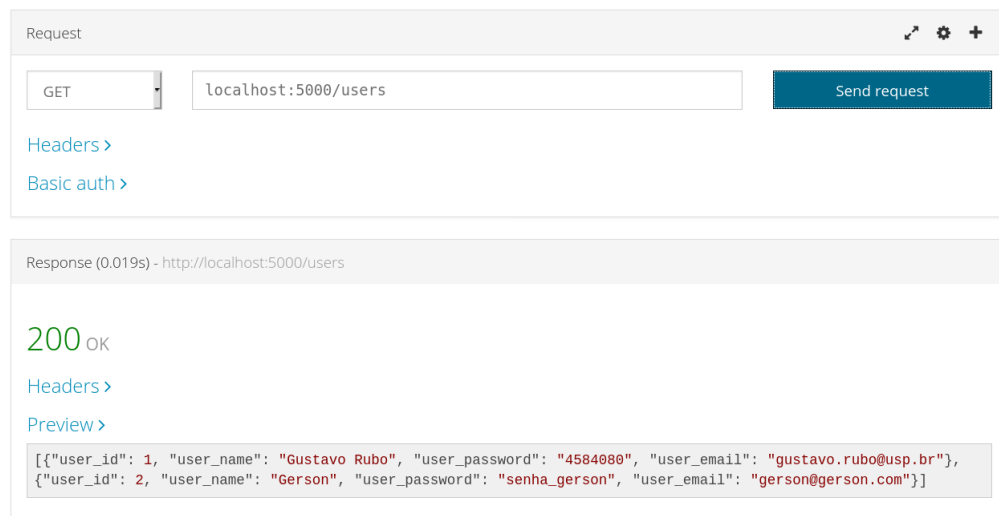


Figura 2: GET /users

Request

GET localhost:5000/users/2 [Send request](#)

[Headers >](#)  
[Basic auth >](#)

Response (0.026s) - http://localhost:5000/users/2

200 OK

[Headers >](#)  
[Preview >](#)

```
{"user_id": 2, "user_name": "Gerson", "user_password": "senha_gerson", "user_email": "gereson@gereson.com"}
```

Figura 3: GET /users/2

Request

PUT localhost:5000/users/1 [Send request](#)

[Headers >](#)  
[Basic auth >](#)  
[Request body >](#)

Type Custom

```
{  
  "user_name": "Gerson",  
  "user_password": "senha_gerson",  
  "user_email": "gereson@gereson.com"  
}
```

Response (0.017s) - http://localhost:5000/users/1

200 OK

[Headers >](#)  
[Preview >](#)

```
{"user_id": 1, "user_name": "Gerson", "user_password": "senha_gerson", "user_email": "gereson@gereson.com"}
```

Figura 4: PUT /users/1

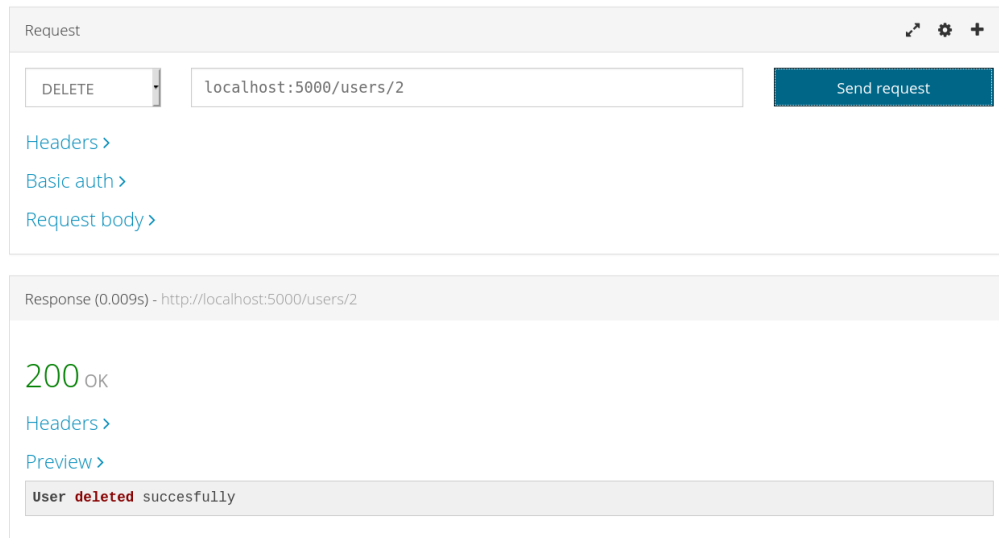


Figura 5: DELETE /users/2

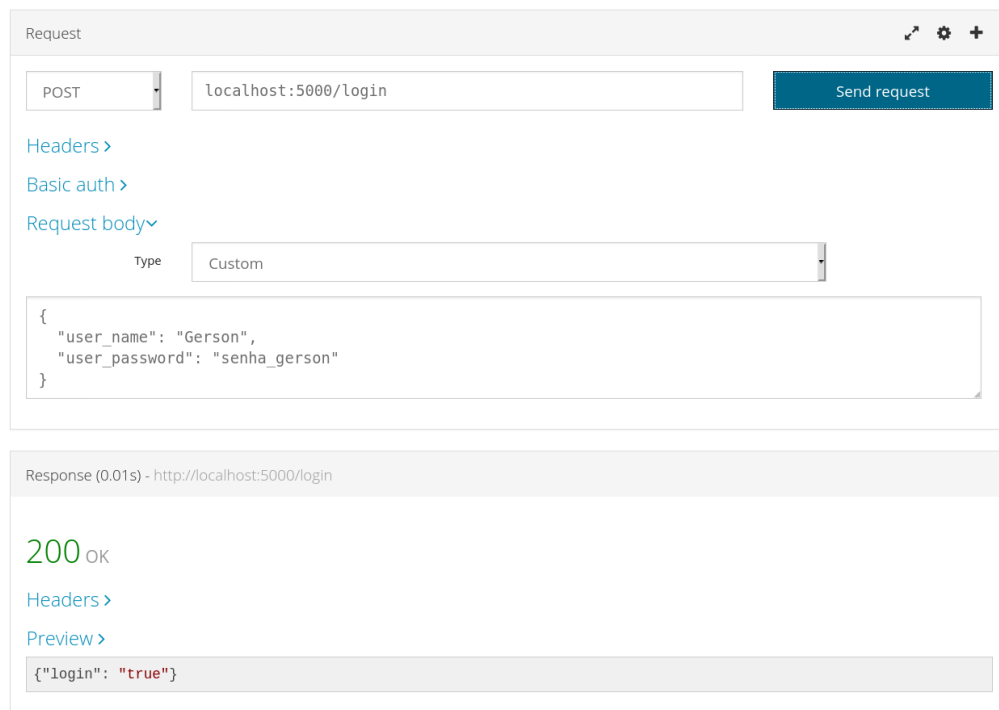


Figura 6: POST /login (com senha correta)

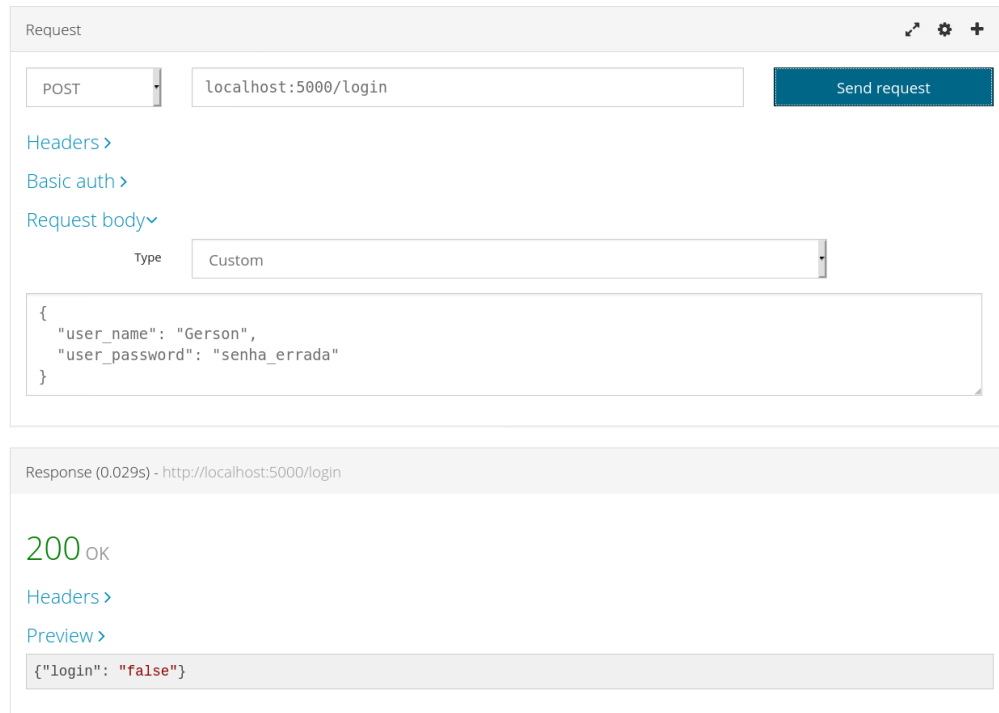


Figura 7: POST /login (com senha incorreta)

## 1.2 Hashing de senhas

Foi feito um servidor que gerencia um banco de dados de usuários como o anterior, porém agora as senhas não são armazenadas em sua forma pura, são criptografadas usando a biblioteca `cryptography` do Python.

A senha "hardcoded" do usuário inicial é criptografada. As senhas de usuários que são adicionados com o POST e modificados com o PUT também são criptografadas. Para todos esses casos, usamos o mesmo sal, que é definido na inicialização do servidor.

As senhas armazenadas tem o tipo de variável "bytes", e portanto não são "JSON serializable". Por isso, as respostas do servidor não são mais convertidas em JSON. É possível escrever uma função de serialização que contorna esse problema, mas não vale o esforço.

A seguir, mostramos a listagem do código.

```
1 from flask import Flask
2 from flask import request
3 import json
4
5 app = Flask(__name__)
6
7 # Inicializando a lista de usuários
8 users = [
9     {
10         "user_id": 1,
11         "user_name": "Gustavo Rubo",
12         "user_password": "4584080",
13         "user_email": "gustavo.rubo@usp.br"
14     }
15 ]
16
17 id_counter = 2
18
19 # Rota para cole o de usuários
20 @app.route("/users", methods=["POST", "GET", "PUT", "DELETE"])
21 def process_users():
22     global id_counter, users
```

```

23     if request.method == "POST":
24         # Criar um usu rio e adicion -lo      lista
25         user = {
26             "user_id": id_counter,
27             "user_name": request.json["user_name"],
28             "user_password": request.json["user_password"],
29             "user_email": request.json["user_email"]
30         }
31         id_counter += 1
32         users.append(user)
33
34         return json.dumps(user), 201
35
36     elif request.method == "GET":
37         # Retornar todos os usu rios
38         return json.dumps(users), 200
39
40     elif request.method == "PUT":
41         return "Method not allowed", 405
42     elif request.method == "DELETE":
43         return "Method not allowed", 405
44
45 # Rota espec fica para usu rios
46 @app.route("/users/<user_id>", methods=["POST", "GET", "PUT", "DELETE"])
47 def process_users_id(user_id):
48     global id_counter, users
49     if request.method == "POST":
50         return "Method not allowed", 405
51
52     elif request.method == "GET":
53         # Retornar o usu rio com o id especificado
54         user = next((u for u in users if u["user_id"] == int(user_id)), None)
55
56         if user:
57             return json.dumps(user), 200
58         else:
59             return "User not found", 404
60
61     elif request.method == "PUT":
62         # Modificar o usu rio com o id especificado
63
64         user = next((u for u in users if u["user_id"] == int(user_id)), None)
65
66         if user:
67             user["user_name"] = request.json["user_name"]
68             user["user_password"] = request.json["user_password"]
69             user["user_email"] = request.json["user_email"]
70             return json.dumps(user), 200
71         else:
72             return "User not found", 404
73
74     elif request.method == "DELETE":
75         # Remover o usu rio com o id especificado
76         user = next((u for u in users if u["user_id"] == int(user_id)), None)
77
78         if user:
79             users[:] = [u for u in users if u["user_id"] != int(user_id)]
80             return "User deleted succesfully", 200
81         else:
82             return "Usu rio n o encontrado", 404
83
84 # Rota para login de usu rio
85 @app.route("/login", methods=["POST"])
86 def process_login():
87     global users
88     if request.method == "POST":
89         user_name = request.json["user_name"]
90         user_password = request.json["user_password"]

```

```

91     user = next((u for u in users if u["user_name"] == user_name), None)
92
93     if user["user_password"] == user_password:
94         return json.dumps({"login": "true"}), 200
95     else:
96         return json.dumps({"login": "false"}), 200

```

Os prints a seguir mostram a criação e login de um usuário com nome "Gerson" e senha 123456.

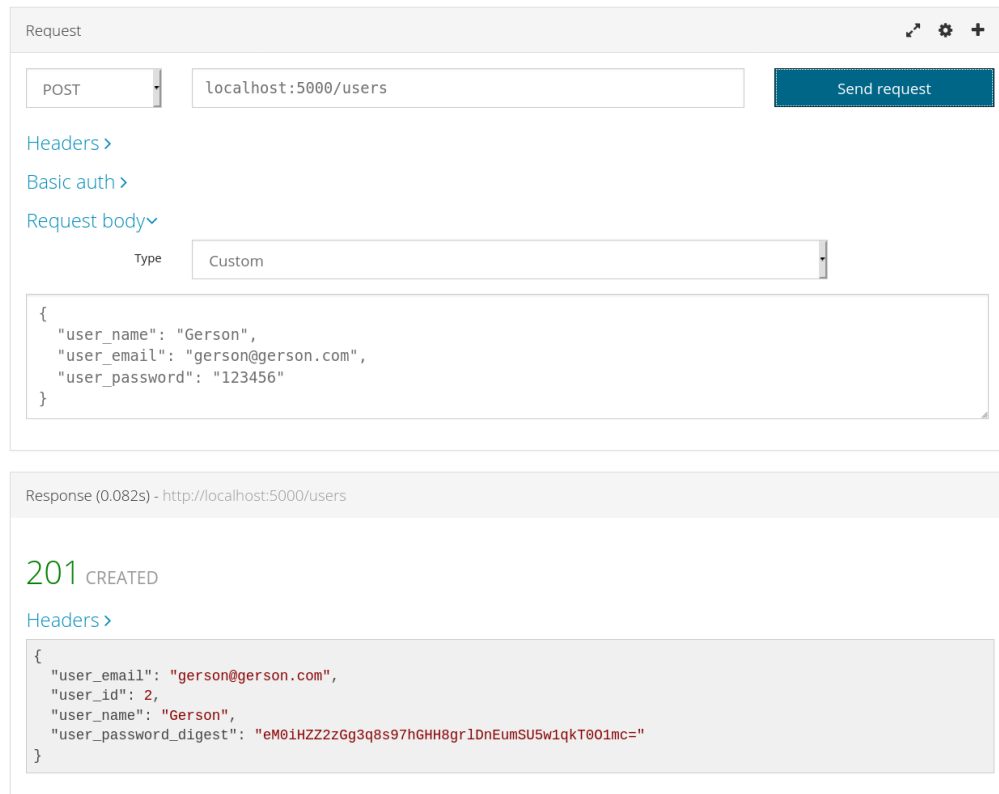


Figura 8: Criação do usuário.

```

New digest password: b'eM0iHZZ2zGg3q8s97hGHH8grlDnEumSU5w1qkT001mc='
127.0.0.1 - - [22/Dec/2020 02:08:27] "POST /users HTTP/1.1" 201 -

```

Figura 9: O resultado do digest da senha "123456".



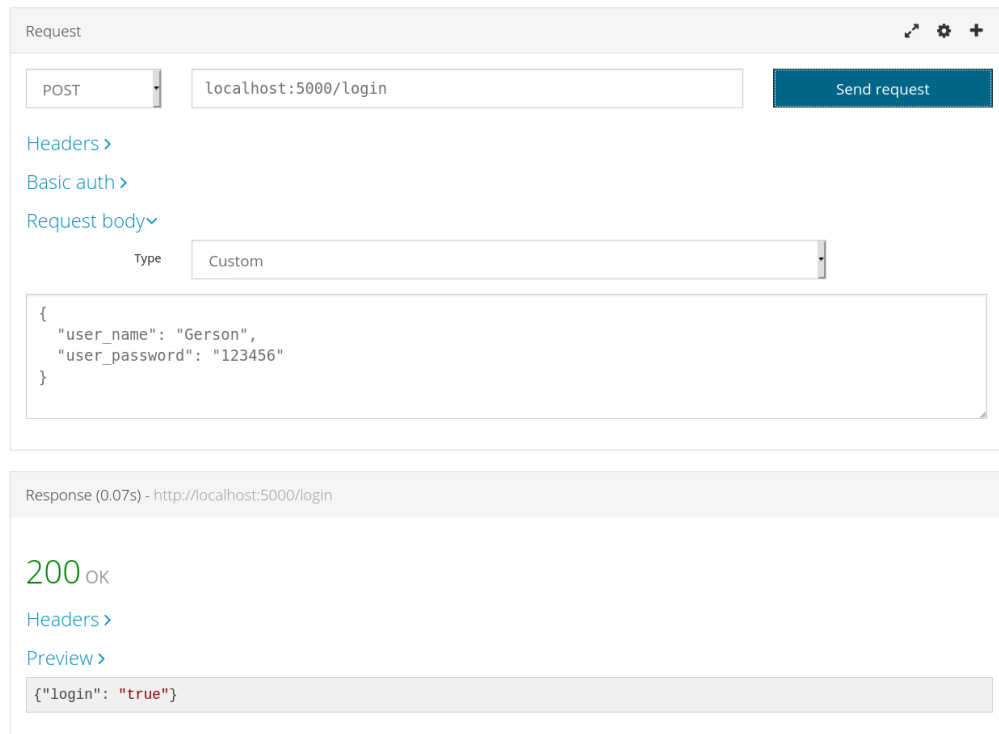


Figura 10: Login efetuado com sucesso.

Como hashes são operações que sempre tem a mesma saída para cada entrada, pode-se criar um banco de dados para senhas comuns. Então, caso sua senha seja uma senha comum (como *password*), ela é vulnerável a um **ataque de dicionário**.

O site Have I been Pwned? tem um registro de mais de 10 bilhões de contas comprometidas. Dessas contas, mais de 3,8 milhões usaram a senha *password*, e 24 milhões usaram a senha *123456*.

Podemos usar o site MD5 Reverse para obter uma senha comum dessas a partir do hash vazado.

### 1.3 Login com Cookie

Para essa parte do exercício, fazemos com que o servidor armazene um cookie de usuário no navegador do cliente, assim fazendo com que o login do usuário seja lembrado e mantido através de várias páginas.

O RESTED Client não tem a funcionalidade de edição de cookies, então a partir de agora o Postman será usado.

No código, a rota de login foi editada, e duas rotas (de boas vindas e de logout) foram criadas. Apenas essas três rotas serão mostradas a seguir.

```

1 # Rota para login de usu rio
2 @app.route("/login", methods=["POST", "GET"])
3 def process_login():
4     global users
5     if request.method == "POST":
6
7         user_name = request.json["user_name"]
8         user_password = request.json["user_password"]
9         user = next((u for u in users if u["user_name"] == user_name), None)
10        kdf = Scrypt(salt=salt, length=32, n=2**14, r=8, p=1, backend=default_backend())
11
12        try:
13            # Caso o nome de usu rio e senha existam no banco de dados
14            kdf.verify(str.encode(user_password), base64.b64decode(user["
15            user_password_digest"]))

```

```

16         resp = make_response(f'Signing in as {request.json["user_name"]}')
17         resp.set_cookie("user_name", request.json["user_name"])
18         return resp
19
20     except cryptography.exceptions.InvalidKey:
21         # Caso de senha inv lida
22         return json.dumps({"message": "Invalid password", "login": "false"}), 200
23     except:
24         # Outros casos (ex: usu rio inexistente)
25         return json.dumps({"message": "Invalid user/password", "login": "false"}), 200
26
27     elif request.method == "GET":
28         user_name = request.cookies.get("user_name")
29
30         if (user_name):
31             return f'Logged in as {user_name}'
32         else:
33             return f'Not logged in yet'
34
35 # Rota para p gina de boas vindas. Usu rio deve estar logado para acessar.
36 @app.route("/welcome", methods=["GET"])
37 def process_welcome():
38
39     if request.method == "GET":
40         user_name = request.cookies.get("user_name")
41
42         if (user_name):
43             return f'Welcome {user_name}'
44         else:
45             return f'You need to log in to access this page.'
46
47 # Rota para logout. Apaga os cookies de usu rio.
48 @app.route("/logout", methods=["DELETE"])
49 def process_logout():
50
51     if request.method == "DELETE":
52         resp = make_response("Cookie removed")
53         resp.set_cookie("user_name", "", max_age=0)
54         return resp

```

Em seguida, verificamos o funcionamento do servidor fazendo um login, conferindo nossa página de boas vindas, fazendo o logout, e conferindo nossa página de boas vindas novamente.

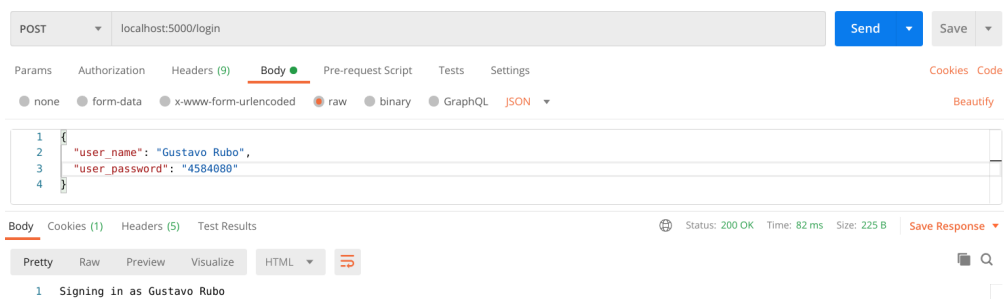


Figura 11: Login efetuado com sucesso.

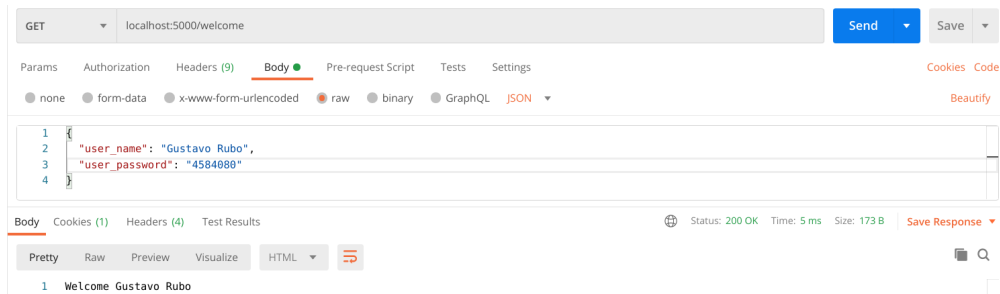


Figura 12: Página de boas vindas para usuário logado.

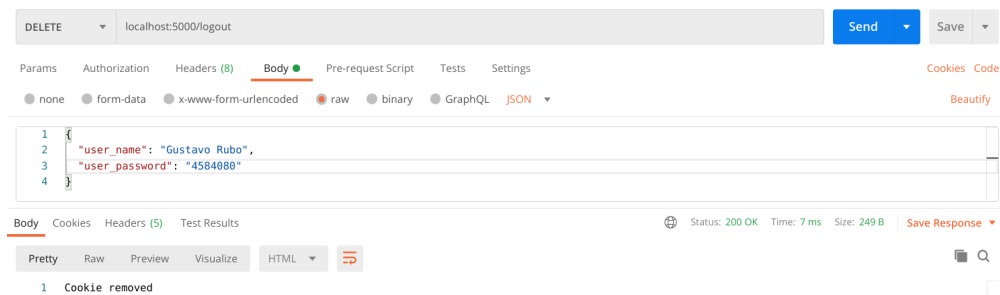


Figura 13: Logout efetuado (Cookies de usuário deletados)

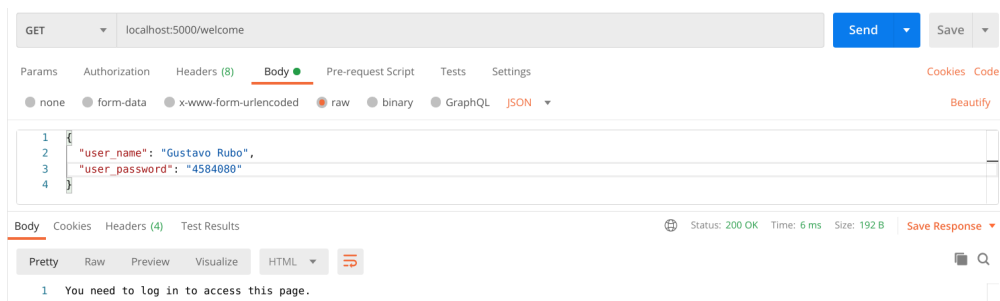


Figura 14: Página de boas vindas para usuário não logado

Os cookies que estamos armazenando por enquanto não são criptografados. Por isso, o sistema é vulnerável a uma alteração manual de cookies.

Com um usuário logado, podemos conferir no Postman qual é o formato de cookies de usuário (Figura [15]). Então, com o postman podemos criar um cookie como ele com o nome do usuário que desejarmos (Figuras 16, 17).

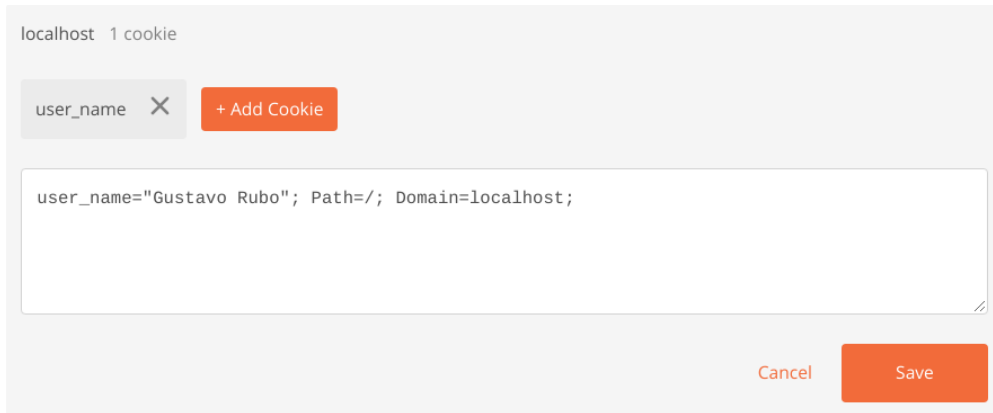


Figura 15: Podemos ver o formato dos cookies de um usuário logado.



Figura 16: Adicionando cookies para o domínio localhost.



Figura 17: Criando um cookie malicioso.

Com o ataque mostrado acima, podemos acessar a página de boas vindas de um usuário sabendo apenas o nome de usuário dele.

## 1.4 Login com Cookie criptografado

Para finalizar o exercício 1, fizemos o servidor enviar cookies criptografados para o cliente. A criptografia é simétrica, e apenas o servidor pode criptografar e descriptografar os cookies. A mensagem criptografada é a junção de uma string *counter*, gerada aleatoriamente e hardcoded no código, com o id do usuário.

Os valores de *key*, *nonce* e *counter* foram gerados com python, como mostrado a seguir.

```

>>> base64.b64encode(os.urandom(32))
b'NbUQlSIhq/Tlq8GuN0LKkMHl0IPvMwOS4ajBWRbom38='
>>> base64.b64encode(os.urandom(16))
b'QwxHQWlXtn6Smq9ajeJlzQ=='
>>> base64.b64encode(os.urandom(16))
b'E9RVpfQ2N0A6rIXX1Nw/7A=='
>>>

```

Figura 18: Gerando valores de key, nonce e counter.

Segue a listagem completa do código final. A classe *EncryptionManager* foi feita de acordo com o exemplo do enunciado.

```

1 import os
2 import base64
3 import cryptography
4 from cryptography.hazmat.primitives.kdf.scrypt import Scrypt
5 from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
6 from cryptography.hazmat.backends import default_backend
7
8 from flask import Flask
9 from flask import request
10 from flask import make_response
11 import json
12
13 counter = 'E9RVpfQ2N0A6rIXX1Nw/7A=='
14
15 class EncryptionManager:
16     def __init__(self):
17         key = base64.b64decode(b'NbUQlSIhq/Tlq8GuN0LKkMHl0IPvMwOS4ajBWRbom38=')
18         nonce = base64.b64decode(b'QwxHQWlXtn6Smq9ajeJlzQ==')
19         aes_context = Cipher(algorithms.AES(key), modes.CTR(nonce), backend=default_backend
20         ())
21         self.encryptor = aes_context.encryptor()
22         self.decryptor = aes_context.decryptor()
23
24     def updateEncryptor(self, plaintext):
25         return self.encryptor.update(plaintext)
26
27     def finalizeEncryptor(self):
28         return self.encryptor.finalize()
29
30     def updateDecryptor(self, ciphertext):
31         return self.decryptor.update(ciphertext)
32
33     def finalizeDecryptor(self):
34         return self.decryptor.finalize()
35
36 # Definir o sal. Enquanto o servidor rodar, ele s ir usar este sal.
37 salt = os.urandom(16)
38 kdf = Scrypt(salt=salt, length=32, n=2**14, r=8, p=1, backend=default_backend())
39
40 app = Flask(__name__)
41
42 # Inicializando a lista de usu rios
43 users = [
44     {
45         "user_id": 1,
46         "user_name": "Gustavo Rubo",
47         "user_password_digest": base64.b64encode(kdf.derive(b"4584080")),
48         "user_email": "gustavo.rubo@usp.br"
49     }
50 ]
51

```

```

52 id_counter = 2
53
54 # Rota para coleção de usuários
55 @app.route("/users", methods=["POST", "GET", "PUT", "DELETE"])
56 def process_users():
57     global id_counter, users
58     if request.method == "POST":
59         # Criar um usuário e adicionar à lista
60
61         kdf = Scrypt(salt=salt, length=32, n=2**14, r=8, p=1, backend=default_backend())
62         user = {
63             "user_id": id_counter,
64             "user_name": request.json["user_name"],
65             "user_password_digest": base64.b64encode(kdf.derive(str.encode(request.json["
66             user_password"]))),
67             "user_email": request.json["user_email"]
68         }
69         id_counter += 1
70         users.append(user)
71
72         print("New digest password:", user["user_password_digest"])
73
74         return user, 201
75
76     elif request.method == "GET":
77         # Retornar todos os usuários
78         return {"users": users}, 200
79
80     elif request.method == "PUT":
81         return "Method not allowed", 405
82     elif request.method == "DELETE":
83         return "Method not allowed", 405
84
85 # Rota específica para usuários
86 @app.route("/users/<user_id>", methods=["POST", "GET", "PUT", "DELETE"])
87 def process_users_id(user_id):
88     global id_counter, users
89     if request.method == "POST":
90         return "Method not allowed", 405
91
92     elif request.method == "GET":
93         # Retornar o usuário com o id especificado
94
95         user = next((u for u in users if u["user_id"] == int(user_id)), None)
96
97         if user:
98             return user, 200
99         else:
100             return "User not found", 404
101
102     elif request.method == "PUT":
103         # Modificar o usuário com o id especificado
104
105         user = next((u for u in users if u["user_id"] == int(user_id)), None)
106
107         if user:
108             user["user_name"] = request.json["user_name"]
109             user["user_password_digest"] = base64.b64encode(kdf.derive(request.json["
110             user_password"]))
111             user["user_email"] = request.json["user_email"]
112
113             print("New digest password:", user["user_password_digest"])
114
115             return user, 200
116         else:
117             return "User not found", 404
118
119     elif request.method == "DELETE":

```

```

118     # Remover o usu rio com o id especificado
119
120     user = next((u for u in users if u["user_id"] == int(user_id)), None)
121
122     if user:
123         users[:] = [u for u in users if u["user_id"] != int(user_id)]
124         return "User deleted succesfully", 200
125     else:
126         return "User not found", 404
127
128 # Rota para login de usu rio
129 @app.route("/login", methods=["POST", "GET"])
130 def process_login():
131     global users
132     if request.method == "POST":
133
134         user_name = request.json["user_name"]
135         user_password = request.json["user_password"]
136         user = next((u for u in users if u["user_name"] == user_name), None)
137         kdf = Script(salt=salt, length=32, n=2**14, r=8, p=1, backend=default_backend())
138         enc_manager = EncryptionManager()
139
140         try:
141             # Caso o nome de usu rio e senha existam no banco de dados
142             kdf.verify(str.encode(user_password), base64.b64decode(user["
user_password_digest"])))
143
144             plaintext = counter+str(user["user_id"])
145             ciphertext = enc_manager.updateEncryptor(bytes(str.encode(plaintext)))
146             enc_manager.finalizeEncryptor()
147
148             print("Cookie encriptado:", base64.b64encode(ciphertext))
149
150             resp = make_response(f'Signing in as {request.json["user_name"]}\'')
151             resp.set_cookie("user_cookie", base64.b64encode(ciphertext))
152             return resp
153
154         except cryptography.exceptions.InvalidKey:
155             # Caso de senha inv lida
156             return json.dumps({"message": "Invalid password", "login": "false"}), 200
157         except:
158             # Outros casos (ex: usu rio inexistente)
159             return json.dumps({"message": "Invalid user/password", "login": "false"}), 200
160
161     elif request.method == "GET":
162         user_name = request.cookies.get("user_name")
163
164         if (user_name):
165             return f'Logged in as {user_name}\'
166         else:
167             return f'Not logged in yet\'
168
169 # Rota para p gina de boas vindas. Usu rio deve estar logado para acessar.
170 @app.route("/welcome", methods=["GET"])
171 def process_welcome():
172
173     if request.method == "GET":
174         if(request.cookies.get("user_cookie")):
175
176             enc_manager = EncryptionManager()
177             ciphertext = base64.b64decode(request.cookies.get("user_cookie"))
178             plaintext = str(enc_manager.updateDecryptor(ciphertext))
179             enc_manager.finalizeDecryptor()
180
181             print("Cookie encriptado:", base64.b64encode(ciphertext))
182
183             user_id = int(plaintext[26:][-1])
184             user = next((u for u in users if u["user_id"] == user_id), None)

```

```

185         return f'Welcome {user["user_name"]}'
186     else:
187         return f'You need to log in to access this page.'
188
189 # Rota para logout. Apaga os cookies de usu rio.
190 @app.route("/logout", methods=["DELETE"])
191 def process_logout():
192     if request.method == "DELETE":
193         resp = make_response("Cookie removed")
194         resp.set_cookie("user_cookie", "", max_age=0)
195         return resp

```

Os cookie são encriptados no servidor quando o usuário faz o login, e são decriptados também no servidor quando o usuário entra na página welcome. A imagem a seguir mostra o cookie encriptado no momento do login e do welcome, respectivamente.

```

Cookie encriptado: b'Nlh9tAdfDvkHlwQ0LoLU05sZ7ZRgSWXuJA=='
127.0.0.1 - - [23/Dec/2020 12:30:37] "POST /login HTTP/1.1" 200 -

Cookie encriptado: b'Nlh9tAdfDvkHlwQ0LoLU05sZ7ZRgSWXuJA=='
127.0.0.1 - - [23/Dec/2020 12:30:46] "GET /welcome HTTP/1.1" 200 -

```

Figura 19: Cookie encriptado.

E enfim, para demonstrar o funcionamento correto do servidor, fazemos uma sequência de login, página de welcome, logout e página de welcome

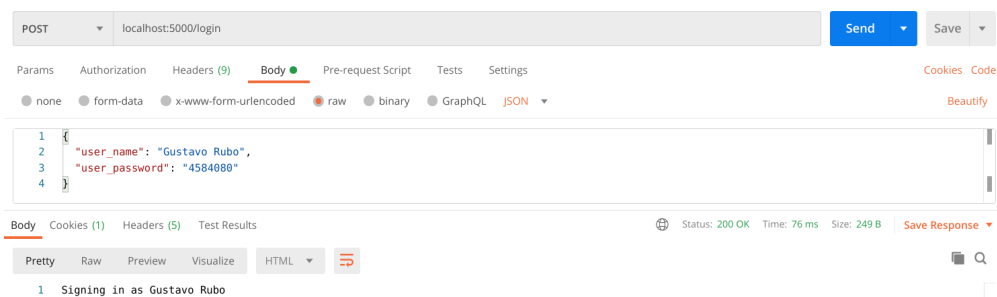


Figura 20: Login efetuado com sucesso.

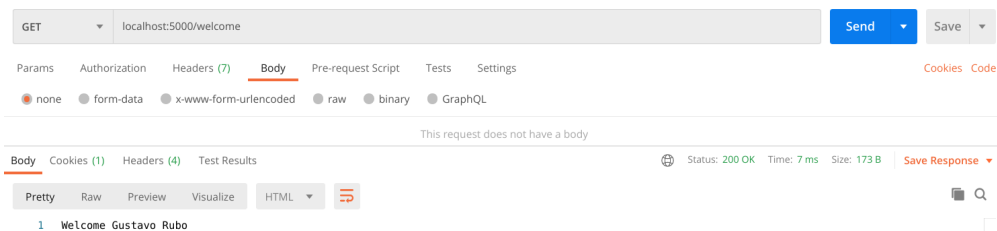


Figura 21: Página de boas vindas para usuário logado.



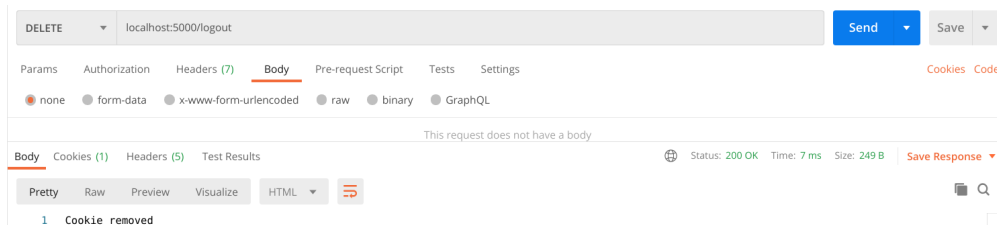


Figura 22: Logout efetuado (Cookies de usuário deletados)

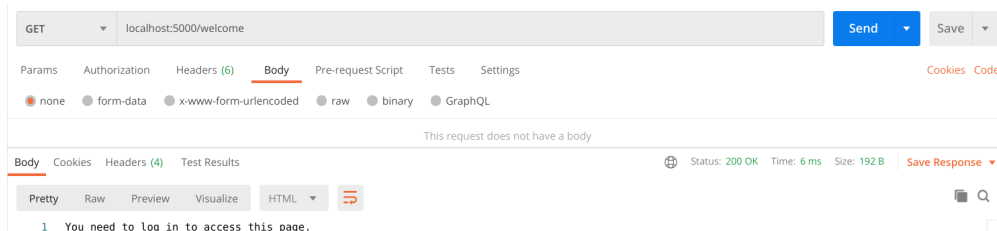


Figura 23: Página de boas vindas para usuário não logado

Caso não concatenarmos o *counter* aos ids, temos um resultado de textos criptografados como o seguinte:

```
Plaintext: b'1' Cipher (hex): 42
Plaintext: b'2' Cipher (hex): 53
Plaintext: b'3' Cipher (hex): 1c
Plaintext: b'4' Cipher (hex): d6
Plaintext: b'5' Cipher (hex): 42
```

Figura 24: Ids criptografados sem a concatenação do counter.

Do jeito que o counter está sendo utilizado neste programa (sendo concatenado ao plaintext), não encontrei que tipo de vulnerabilidade pode existir caso omitirmos ele. Inclusive, o livro texto não mostra uma string sendo contatenada a mensagens no diagrama da encriptação CTR.

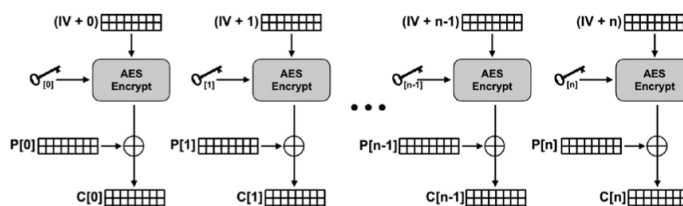


Figura 25: Esquema da encriptação CTR.

Caso um atacante conseguir interceptar a mensagem criptografada do id, sabendo qual é o id, ele pode conseguir informações sobre a chave utilizada para criptografá-lo, mas isso não deve ser um problema se a mesma chave nunca for repetida.

## 2 Exercício 2 - Aplicativo de troca de mensagens

Não feito.

## 3 Considerações sobre os exercícios

Ter os enunciados separados em requisitos funcionais e lista de coisas a apresentar no relatório é muito conveniente e fácil de seguir, não ficam dúvidas sobre o que temos que fazer.

Por ter uma quantidade de trabalho muito grande de programação, muitas vezes eu estava só fazendo o programa sem pensar na teoria do que eu estava fazendo, só colocando os exemplos no lugar certo e fazendo funcionar. Só as questões conceituais me fizeram revisar os slides e assistir pedaços das aulas de novo.

Além disso, confesso que é difícil ter motivação para completar todo o exercício tendo já passado da matéria com a nota dos anteriores, ainda mais com o saco cheio de final de ano. Tendo, dito isso, boas festas!