

# Лабораторна робота 1

## Розробка програми розмежування повноважень користувачів на основі парольної автентифікації

Приходько Юрій ФБ-12, Варіант 13

### Мета роботи:

Реалізувати алгоритм розмежування користувачів за допомогою парольної автентифікації.

### Вміст завдання

Необхідно розробити додаток, який відповідає наступним вимогам:

1. Програма повинна забезпечувати роботу в двох режимах: адміністратора (користувача з фіксованим ім'ям ADMIN) і звичайного користувача.
2. У режимі адміністратора програма повинна підтримувати наступні функції (при правильному введенні пароля):
  - зміна пароля адміністратора (при правильному введенні старого пароля);
  - перегляд списку імен зареєстрованих користувачів і встановлених для них параметрів (блокування облікового запису, включення обмежень на вибрані паролі) – всього списку цілком в одному вікні або по одному елементу списку з можливістю переміщення до його початку або кінця;
  - додавання унікального імені нового користувача до списку з порожнім паролем (рядком нульової довжини);
  - блокування можливості роботи користувача із заданим ім'ям;
  - включення або відключення обмежень на вибрані користувачем паролі (відповідно до індивідуального завдання, визначуваного номера варіанту);
  - завершення роботи з програмою.
3. У режимі звичайного користувача програма повинна підтримувати лише функції зміни пароля користувача (при правильному введенні старого пароля) і завершення роботи, а всі останні функції мають бути заблоковані.
4. За відсутності введеного у вікні входу імені користувача в списку зареєстрованих адміністратором користувачів програма повинна видавати відповідне повідомлення і надавати користувачеві можливість повторного введення імені або завершення роботи з програмою.

5. При неправильному введенні пароля програма повинна видавати відповідне повідомлення і надавати користувачеві можливість повторного введення. При трикратному введенні невірної пароля робота програми повинна завершуватися.
6. При заміні пароля програма повинна просити користувача підтвердити введений пароль шляхом його повторного введення.
7. Якщо вибраний користувачем пароль не відповідає необхідним обмеженням (при установці відповідного параметра облікового запису користувача), то програма повинна видавати відповідне повідомлення і надавати користувачеві можливість введення іншого пароля, завершення роботи з програмою (при першому вході даного користувача) або відмови від зміни пароля.
8. Інформація про зареєстрованих користувачів, їх паролі, відсутність блокування їх роботи з програмою, а також включенні або відключенні обмежень на вибрані паролі повинна зберігатися в спеціальному файлі. При першому запуску програми цей файл повинен створюватися автоматично і містити інформацію лише про адміністратора, що має порожній пароль.
9. Інтерфейс з програмою має бути організований на основі меню, обов'язковою частиною якого має бути підміню «Довідка» з командою «Про програму». При виборі цієї команди повинна видаватися інформація про автора програми і видане індивідуальне завдання. Інтерфейс користувача програми може також включати панель управління з дублюючими команди меню графічними кнопками і рядок стану.

## Особистий варіант (обмеження на вибрані паролі) №13

Наявність букв, розділового і знаків арифметичних операцій знаків

### Хід роботи

У ході виконання лабораторної роботи, було визначено функціонал додатку наступним чином. Спілкування з користувачем буде зроблено за допомогою графічного додатку (GUI), а вся логіка програми що керує її функціоналом має бути винесена у окремі контролери для полегшення розробки та розуміння вихідного коду програми.

Зберігання інформації про користувача, включно з ім'ям, паролем, роллю (адмін чи звичайний користувач), а також відомостей про стан обмеження пароля користувача, стан блокування його акаунту і кількість неправильних спроб вводу пароля, будуть зберігатись у окремому файлі у форматі `json`.

Пароль має зберігатись у хешованому вигляді для забезпечення безпеки користувачів при компрометації файлу з даними. Для хешування був обраний алгоритм `argon2`.

Логування подій що відбуваються в програмі можуть бути з поміткою `Error` або `Info` та зберігатись у окремому файлі, також у форматі `json`.

При першому запуску програми мають ініціалізуватись `json` файли, та створюватись єдиний користувач з роллю адміністратора застосунку, та пустим паролем. Хоча в завданні і зазначено при створенні нового користувача надавати йому пустий пароль за замовчуванням, варто зазначити що в реальному застосунку з міркувань безпеки варто створювати користувача з випадковим тимчасовим паролем, і передавати відомість про цей пароль через надійні джерела інформації що не можуть бути скомпроментовані, а після першого логіну примушувати користувача змінити його.

При запуску програми користувач побачить логін форму, де зможе увійти в свій акаунт. У випадку блокування користувача, зміни обмежень на його пароль, або просто при створенні нового акаунту буде встановлений атрибут користувача `force_change_passwd` що змусить користувача змінити пароль при наступній спробі логіну в акаунт. Це здійснено з міркувань безпеки.

В завданні вказано виводити відповідні повідомлення при неправильному вводі юзенеюму чи паролю, проте для протидії атакам `user enumeration` при фазінгу додатку це повідомлення замінено на одне: `Username or/and Password incorrect` .

При введенні неправильного паролю тричі акаунт стає заблокованим, і подальші спроби логіну будуть відкинуті. Розблокувати його може лише адміністратор.

Після успішної автентифікації адміністратора, він має мати змогу в окремому вікні передивитись користувачів, а також змінити стан їх акаунту (заблокувати/розблокувати, дати/забрати обмеження на пароль). Також у цьому вікні можна створити та видалити користувачів. З міркувань безпеки для підтвердження видалення необхідно повторно ввести пароль адміністратора.

Так як додатком не передбачено зміни ролі користувача, чи створення нового користувача адміністратора для підвищення відмовостійності адміністратор не може видалити чи заблокувати свій акаунт.

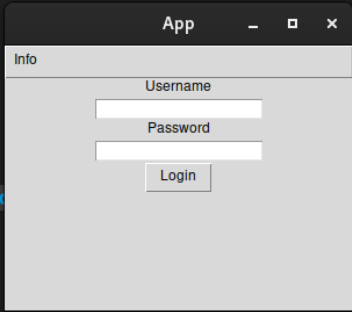
В результаті виконання лабораторної роботи, за допомогою мови `python` було розроблено додаток що імплементує роздуми, описані вище.

Приклад використання додатку, його тестування, а також важливі шматки вихідних кодів надані у вигляді зображень далі. Доступ до вихідних кодів у повному обсязі через `github`, [Посилання](#)

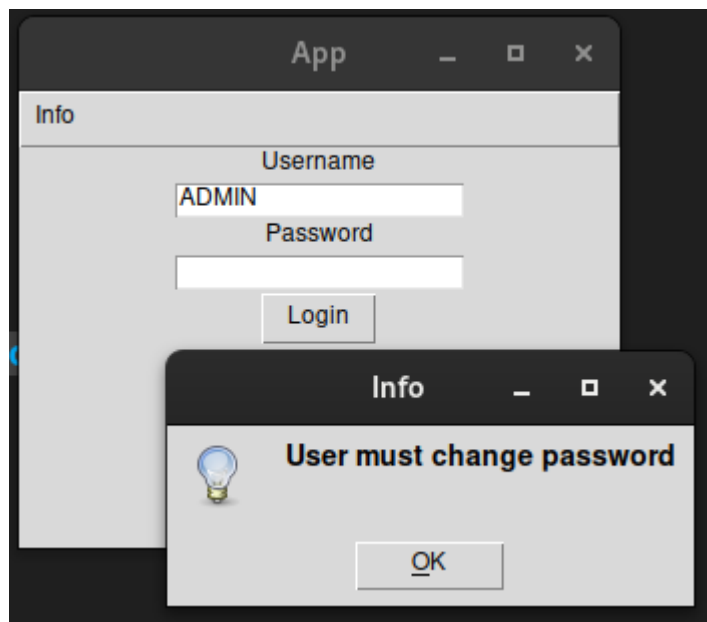
Створення нового файлу `database.json` після першого запуску. Логін форма додатку.

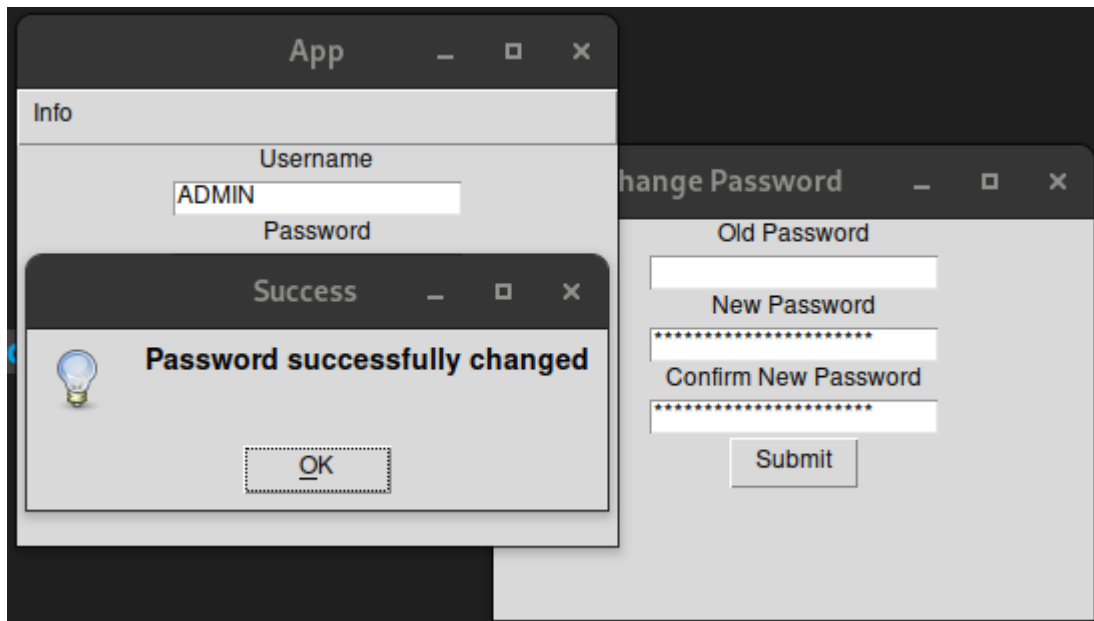
```
^ > ~/Documents/progsec/lab1/sources > main ?1
ls json
database.json logs.json

^ > ~/Documents/progsec/lab1/sources > main ?1
cat json/database.json | jq
{
  "ADMIN": {
    "password": "$argon2id$v=19$m=65536,t=3,p=4$NodMK2Xf9JVoiDtGECpAA$5RPg74/I5hoSgeeqpgytsIlBwMI0jNZDh80ajBGD8dw",
    "role": "admin",
    "restricted": false,
    "force_change_password": true,
    "banned": false,
    "inc_att": 0
  }
}
```

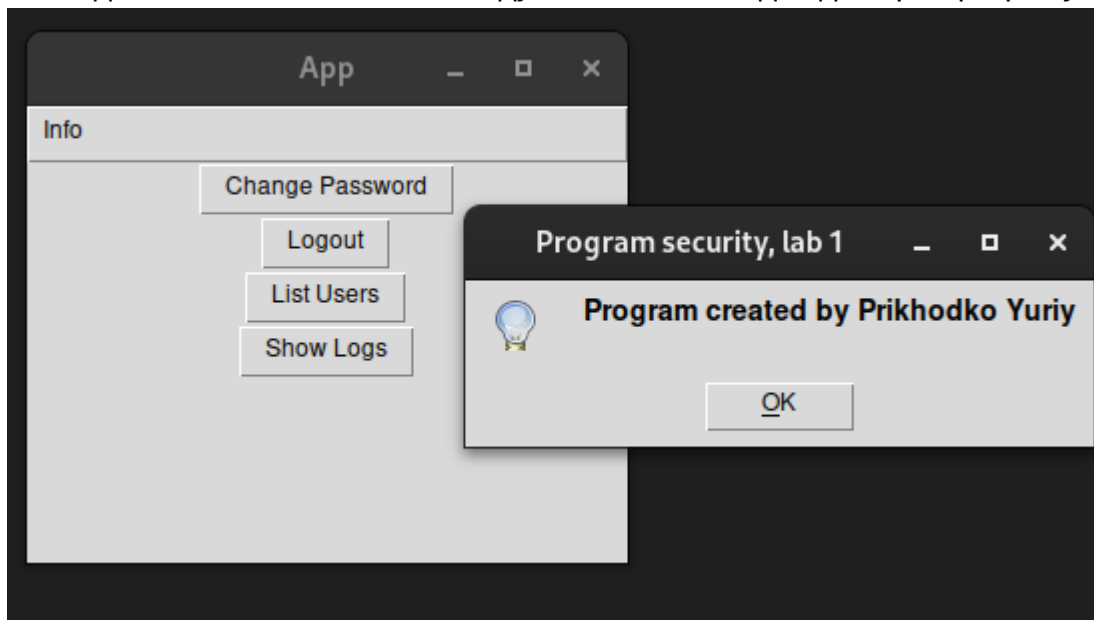


Перший вхід у адміністративний акаунт

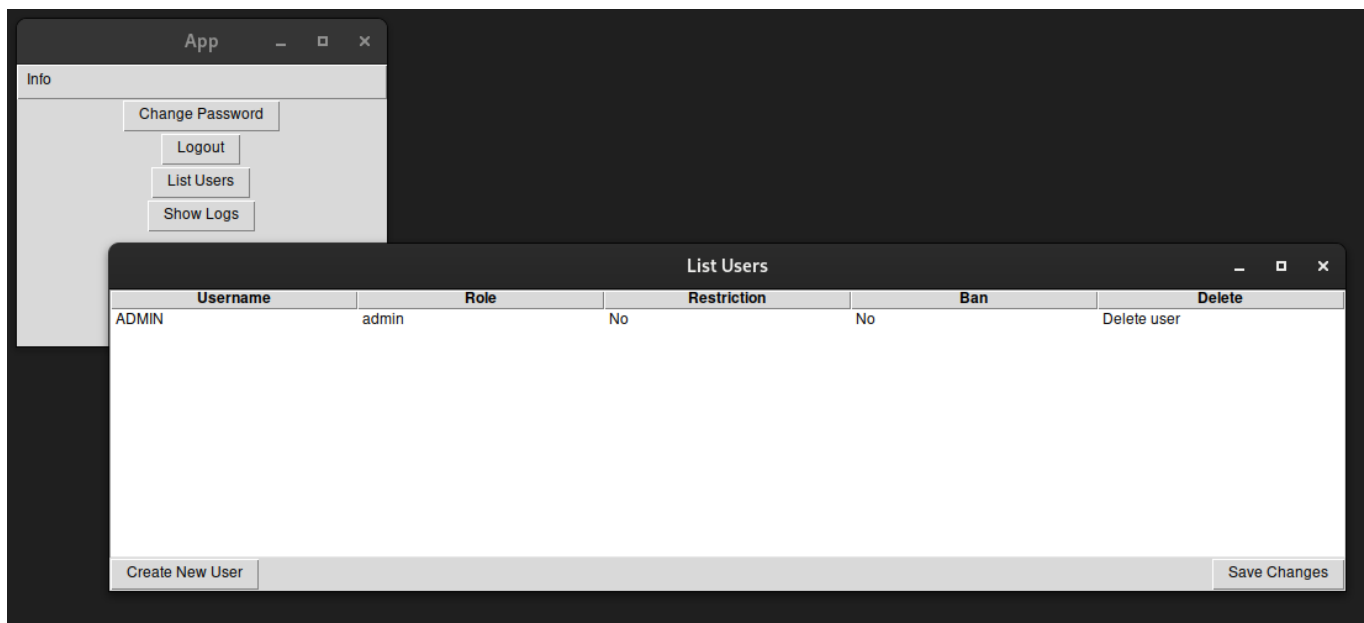




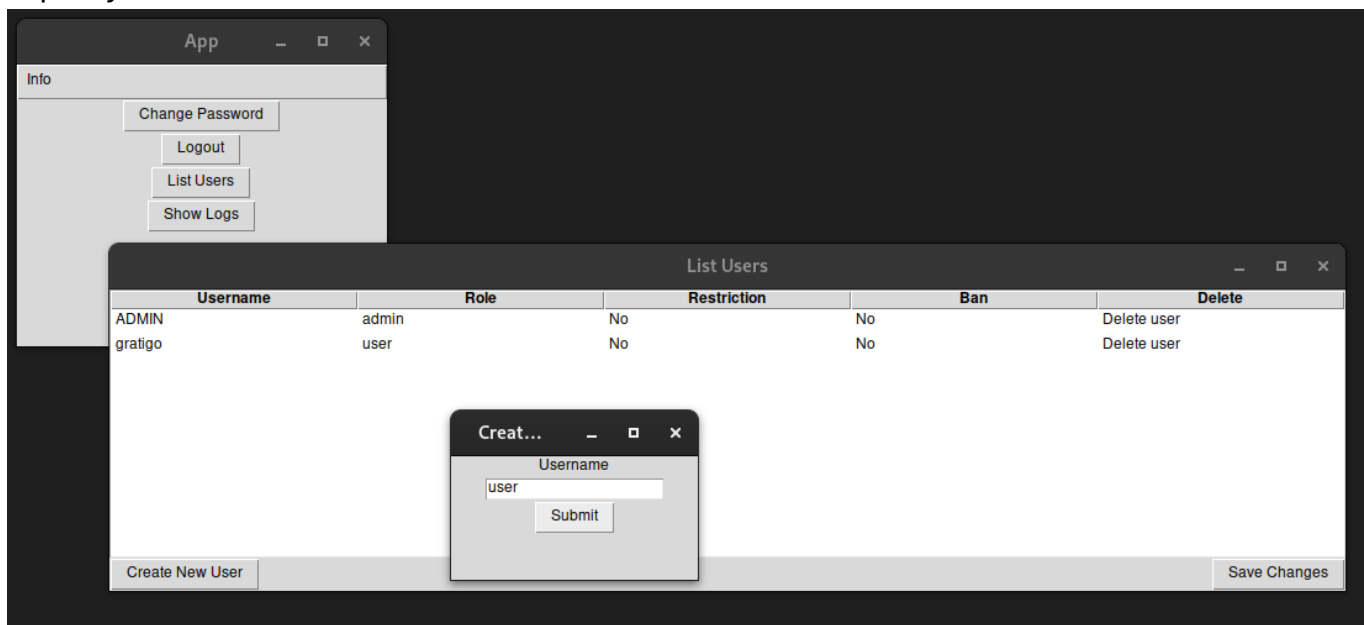
Вигляд основного вікна після входу, а також меню довідки про програму:



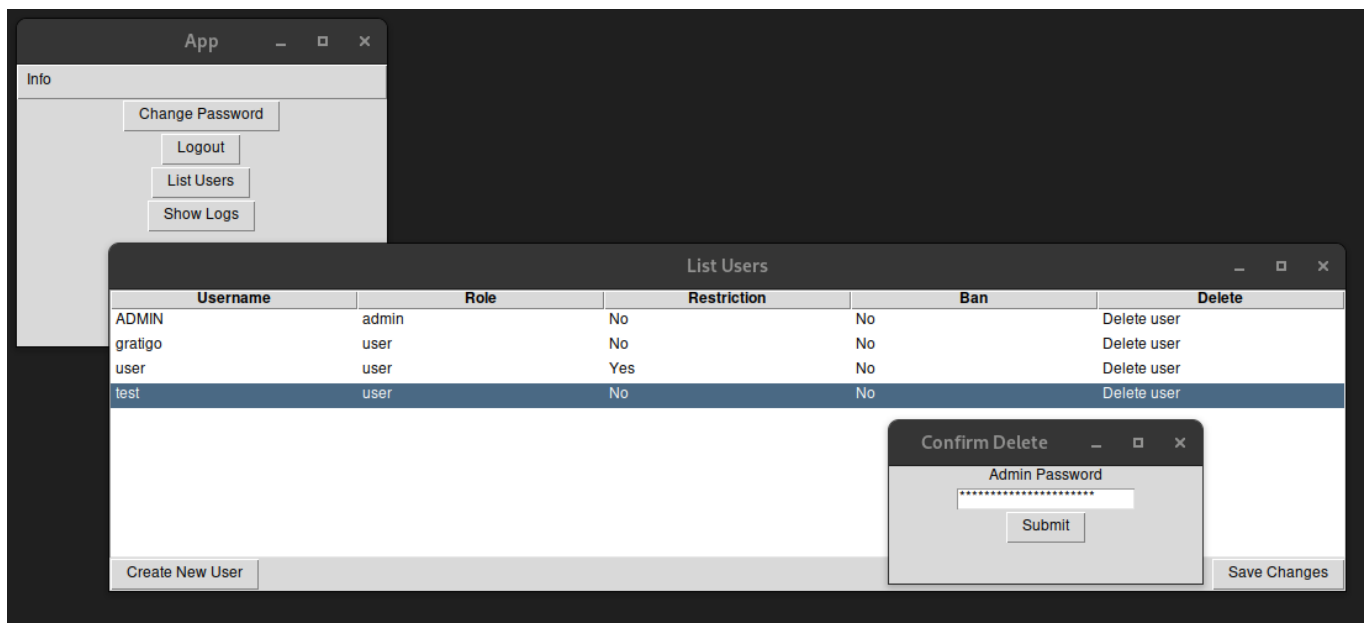
Вікно списку користувачів:



Створимо трьох нових користувачів на зображенні показаний процес створення другого користувача:

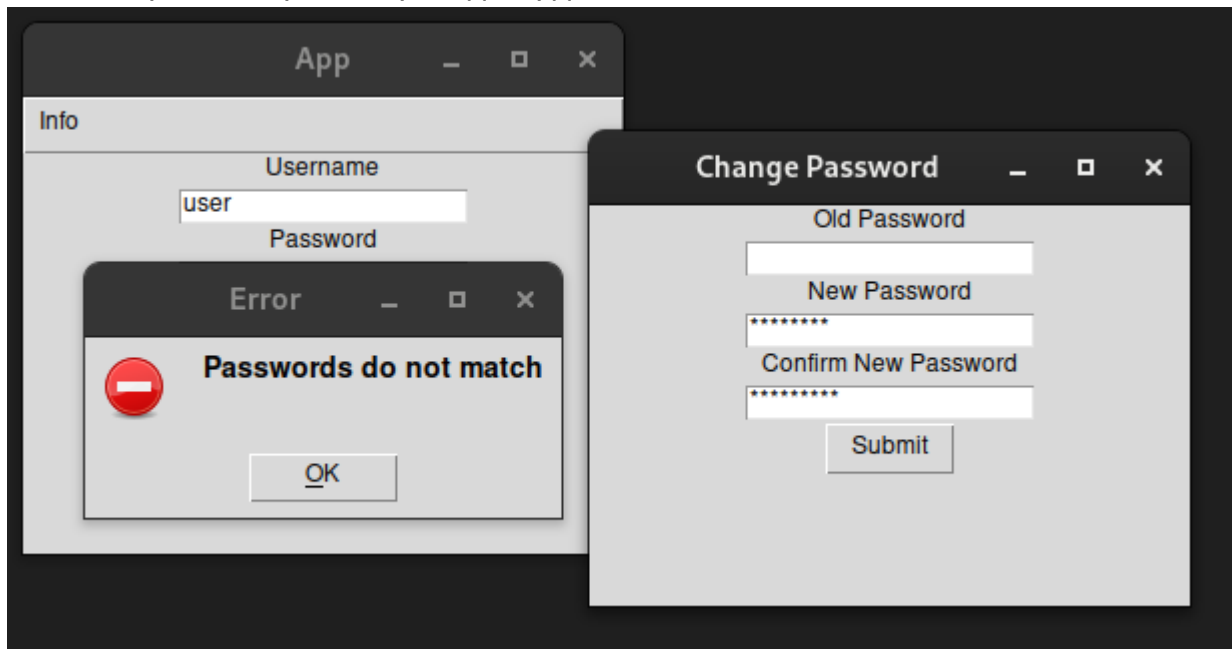


Для створеного користувача `user` поставимо обмеження на пароль, третього користувача `test` видалимо, та збережемо зміни.

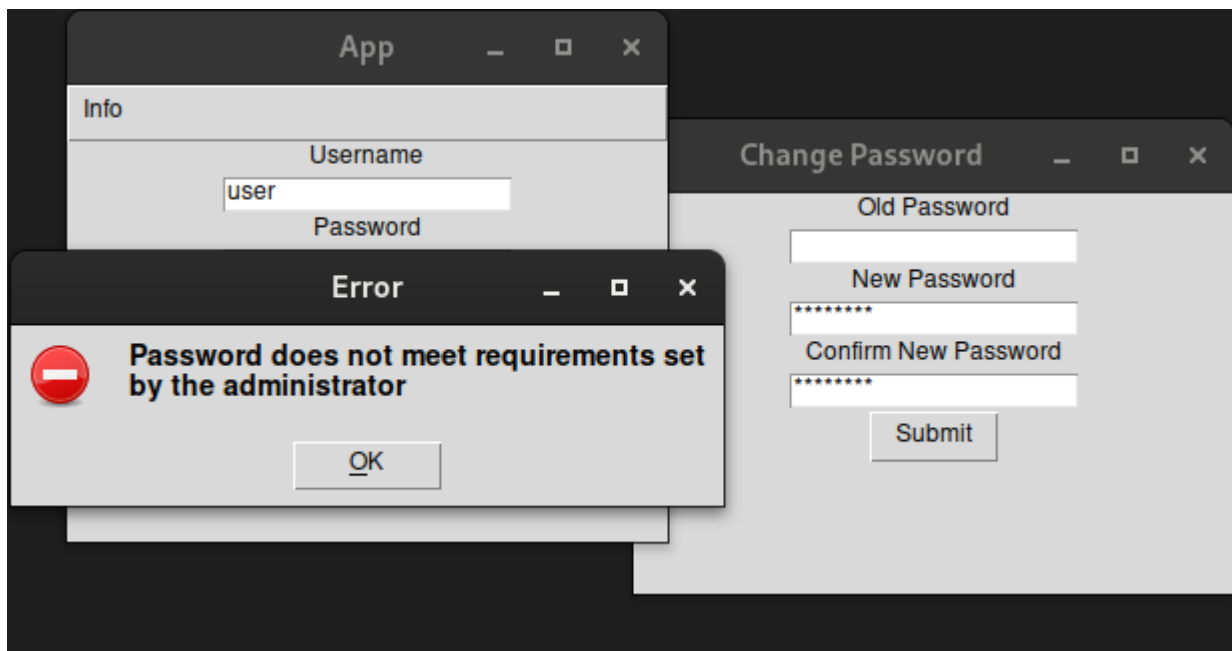


Вийдемо з акаунту адміна та спробуємо залогінитись як інший користувач. При першому вході отримаємо форму зміни паролю на новий, в акаунт не можливо ввійти доки пароль не буде змінений на новий що відповідає вимогам. На прикладі користувача з обмеженням `user` спробуємо створити незадовільний пароль.

Помилка різних паролей при підтвердженні нового:

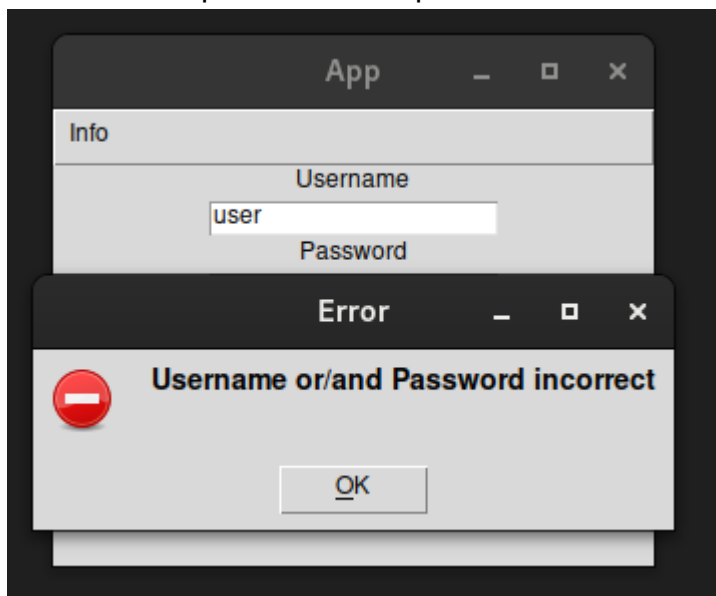


Помилка вимог до паролю:



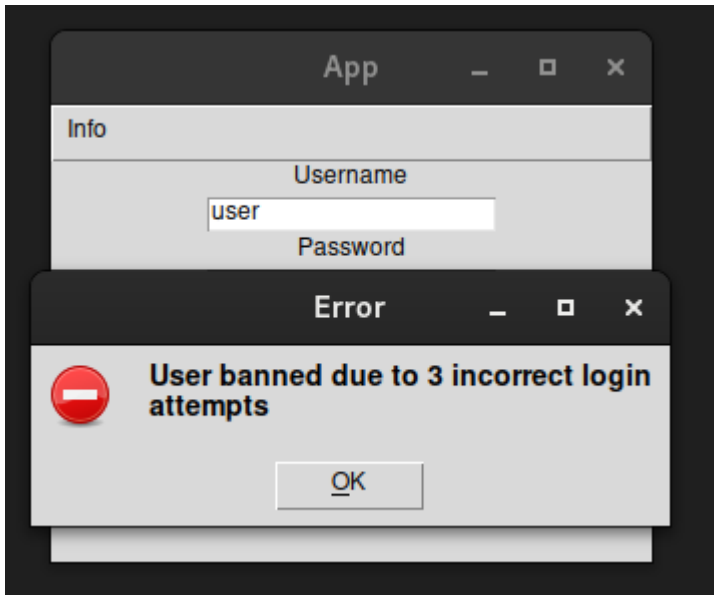
Після зміни паролю на задовільний, спробуємо тричі ввести неправильний пароль до акаунту `user` .

Помилка неправильного паролю:

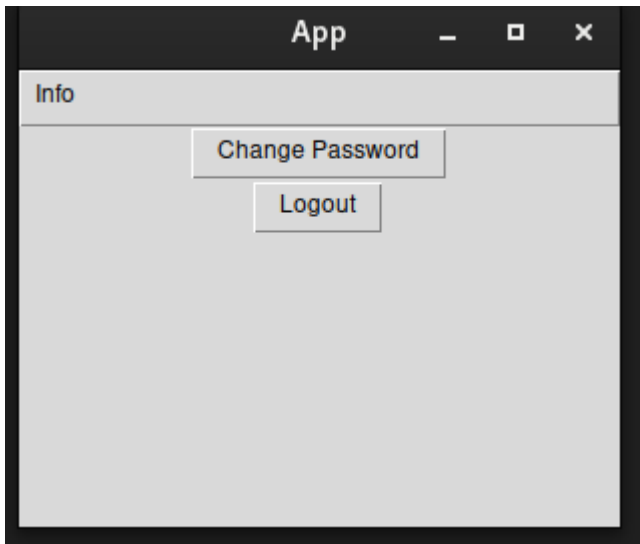




Блокування акаунту після 3 невдалих спроб:



Встановлення паролю при першому вході та вдалий вхід для акаунту `gratigo`, з доступного функціоналу тільки зміна паролю.



Зняти блокування з користувача, а також переглянути логи невдалих спроб можна з акаунту з роллю адміністратора.

Розблокування акаунта через список користувачів:

App

List Users

Username	Role	Restriction	Ban	Delete
ADMIN	admin	No	No	Delete user
gratigo	user	No	No	Delete user
user	user	Yes	Yes	Delete user

Create New User

Save Changes

Перегляд логів додатку з акаунту адміністратора:

App

Info

Change Password

Logout

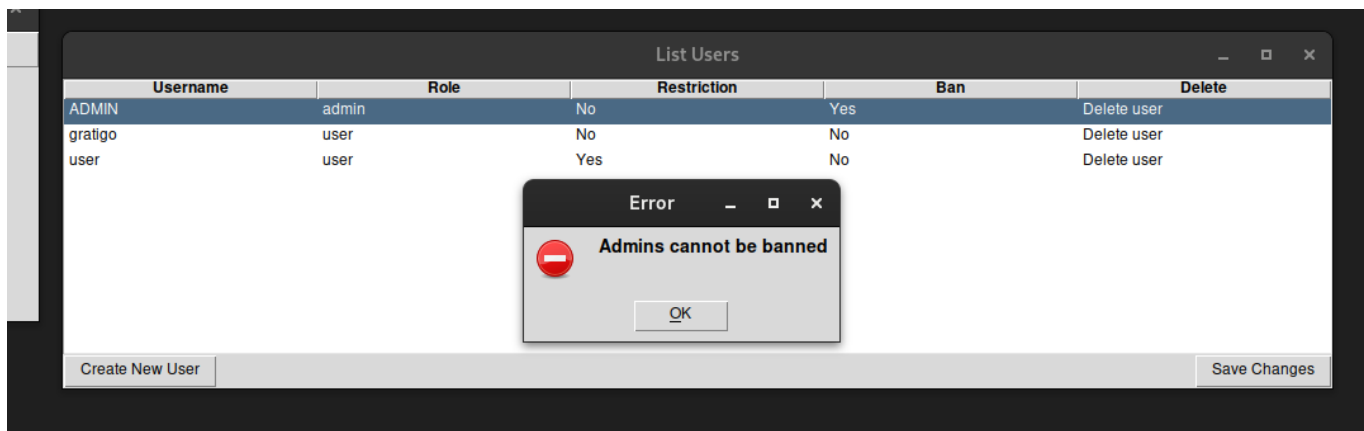
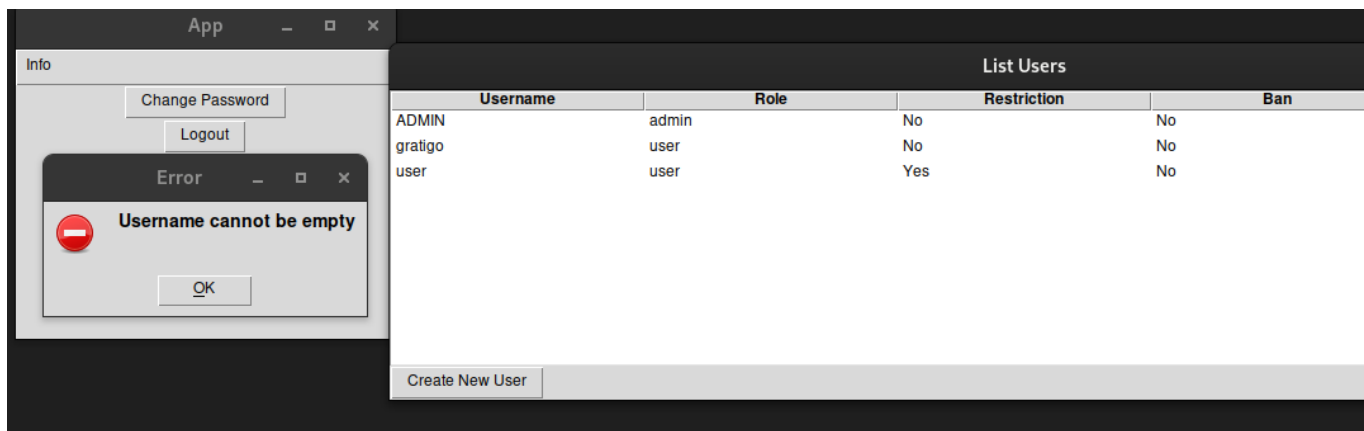
List Users

Show Logs

Logs

Date	Type	Message
2024-05-01 20:07:22	Info	User 'ADMIN' logged in
2024-05-01 20:06:39	Info	User 'gratigo' logged in
2024-05-01 20:06:35	Info	Password for user 'gratigo' changed
2024-05-01 20:06:29	Info	Attempt to login as 'gratigo' failed. Reason: User must change password
2024-05-01 20:06:23	Info	User 'user' ban status changed to False
2024-05-01 20:04:49	Info	User 'ADMIN' logged in
2024-05-01 20:03:41	Error	User 'user' banned, reason: User banned due to 3 incorrect login attempts
2024-05-01 20:03:41	Error	Attempt to login as 'user' failed. Reason: Incorrect Password
2024-05-01 20:03:39	Error	Attempt to login as 'user' failed. Reason: Incorrect Password
2024-05-01 20:03:04	Error	Attempt to login as 'user' failed. Reason: Incorrect Password
2024-05-01 20:03:02	Info	Password for user 'user' changed
2024-05-01 20:01:27	Error	Attempt to change password for restricted user 'user' failed. Reason: Password does not meet requirements set by the administrator
2024-05-01 20:00:40	Info	Attempt to login as 'user' failed. Reason: User must change password
2024-05-01 19:58:37	Info	User 'user' restrictions changed to True
2024-05-01 19:58:35	Info	User 'test' deleted
2024-05-01 19:58:01	Info	User 'test' created
2024-05-01 19:57:14	Info	User 'user' created
2024-05-01 19:56:16	Info	User 'gratigo' created
2024-05-01 19:53:55	Info	User 'ADMIN' logged in
2024-05-01 19:53:17	Info	Password for user 'ADMIN' changed
2024-05-01 19:52:21	Info	Attempt to login as 'ADMIN' failed. Reason: User must change password
2024-05-01 19:43:05	Info	Database file not found, creating new one
2024-05-01 19:43:05	Info	Loading database file

Відповідні попередження при спробі бану адміністративного акаунту, та створенні користувача з пустим іменем.



Вихідні коди додатку у вигляді зображень:

Контроллер що відповідає за процес перевірки паролю при логіні, та його зміни при виклику відповідної форми, `security_controller.py`

```

controllers > security_controller.py > ...
1  from argon2 import PasswordHasher
2  from json import loads, dumps
3  from string import ascii_letters, punctuation
4  from errors import *
5  from controllers.log_controller import *
6
7  ARITHM_SYMBOLS = "+-*/^%"
8  EMPTY_PASS = "$argon2id$v=19$m=65536,t=3,p=4$0Nod8K2Xf9JVoqiDtGECFAA$5Rfg74/I5hoSgeeqpgytsi1BMMi0jNIDh80ajBGD8dw"
9  DB_FILEPATH = 'json/database.json'
10
11 def initDB() -> None:
12     try:
13         log(INFO_LOG, "Loading database file")
14         with open(DB_FILEPATH, 'r') as f:
15             data = loads(f.read())
16     except:
17         log(INFO_LOG, "Database file not found, creating new one")
18         data = {
19             "ADMIN": {
20                 "password": EMPTY_PASS,
21                 "role": "admin",
22                 "restricted": False,
23                 "force_change_password": True,
24                 "banned": False,
25                 "inc_att": 0
26             }
27         }
28         with open(DB_FILEPATH, 'w') as f:
29             f.write(dumps(data))
30
31 def checkPassword(username:str, password:str) -> bool:
32     with open(DB_FILEPATH, 'r') as f:
33         database = loads(f.read())
34
35     ph = PasswordHasher()
36     stored_password = database[username]["password"]
37     try:
38         ph.verify(stored_password, password)
39     except:
40         return False
41     return True
42
43 def checkLogin(username:str, password:str) -> dict:
44
45     ph = PasswordHasher()
46
47     with open(DB_FILEPATH, 'r') as f:
48         database = loads(f.read())
49
50     if username not in database:
51         log(ERR_LOG, f"Attempt to login as '{username}' failed. Reason: {INC_USER_ERR}")
52         return {"Error": INC_LOGIN_ERR}
53
54     if database[username]["banned"]:
55         log(ERR_LOG, f"Attempt to login as '{username}' failed. Reason: {USER_BAN_ERR}")
56         return {"Error": USER_BAN_ERR}
57
58     if database[username]["force_change_password"]:
59         log(INFO_LOG, f"Attempt to login as '{username}' failed. Reason: {CHANGE_PASS_ERR}")
60         user = {"username": username, "role": database[username]["role"]}
61         return {"Info": CHANGE_PASS_ERR, "user": user}
62
63     stored_password = database[username]["password"]
64     try:
65         ph.verify(stored_password, password)
66     except:
67         error = INC_LOGIN_ERR
68         database[username]["inc_att"] += 1
69         log(ERR_LOG, f"Attempt to login as '{username}' failed. Reason: {INC_PASS_ERR}")
70
71         if database[username]["inc_att"] >= 3:
72             database[username]["banned"] = True
73             database[username]["force_change_password"] = True
74             error = ATT_BAN_ERR
75             log(ERR_LOG, f"User '{username}' banned, reason: {ATT_BAN_ERR}")
76
77     with open(DB_FILEPATH, 'w') as f:
78         f.write(dumps(database))
79
80     return {"Error": error}
81
82     log(INFO_LOG, f"User '{username}' logged in")
83     database[username]["inc_att"] = 0
84     with open(DB_FILEPATH, 'w') as f:
85         f.write(dumps(database))
86     return {"username": username, "role": database[username]["role"]}
87

```

```

98 def checkPasswordRestrictions(password:str) -> bool:
99     if (
100         any(char in ascii_letters for char in password) and
101         any(char in punctuation for char in password) and
102         any(char in ARITHM_SYMBOLS for char in password)
103     ):
104         return True
105     return False
106
107 def changePassword(old_password:str, new_password:str, user:dict) -> dict:
108     with open(DB_FILEPATH, 'r') as f:
109         database = loads(f.read())
110
111     if database[user["username"]]["restricted"]:
112         if not checkPasswordRestrictions(new_password):
113             log(ERR_LOG, f"Attempt to change password for restricted user '{user['username']}' failed. Reason: {PASS_RESRT_ERR}")
114             return {"Error": PASS_RESRT_ERR}
115
116     ph = PasswordHasher()
117
118     username = user["username"]
119     stored_password = database[username]["password"]
120     try:
121         ph.verify(stored_password, old_password)
122     except:
123         log(ERR_LOG, f"Attempt to change password for user '{username}' failed. Reason: {INC_PASS_ERR}")
124         return {"Error": INC_PASS_ERR}
125
126     database[username]["password"] = ph.hash(new_password)
127     database[username]["force_change_password"] = False
128     with open(DB_FILEPATH, 'w') as f:
129         f.write(dumps(database))
130
131     log(INFO_LOG, f"Password for user '{username}' changed")
132     return user

```

Контроллер що відповідає за функції доступні адміну (керування користувачами),

# admin\_controller.py

```

1  from json import loads, dumps
2  from errors import *
3  from controllers.log_controller import *
4  from controllers.security_controller import DB_FILEPATH, EMPTY_PASS, checkPassword
5
6  def getUsers() -> dict:
7      with open(DB_FILEPATH, 'r') as f:
8          database = loads(f.read())
9
10         user_info = database.copy()
11         for i in user_info:
12             user_info[i].pop("password")
13             if user_info[i]["restricted"]: user_info[i]["restricted"] = "Yes"
14             else: user_info[i]["restricted"] = "No"
15             if user_info[i]["banned"]: user_info[i]["banned"] = "Yes"
16             else: user_info[i]["banned"] = "No"
17
18         return user_info
19
20
21  def updateUser(username:str, restriction, ban) -> dict:
22      with open(DB_FILEPATH, 'r') as f:
23          database = loads(f.read())
24          old_restriction = database[username]["restricted"]
25          database[username]["restricted"] = True if restriction == "Yes" else False
26          database[username]["force_change_password"] = True if restriction == "Yes" else database[username]["force_change_password"]
27          if old_restriction != database[username]["restricted"]:
28              log(INFO_LOG, f"User '{username}' restrictions changed to {database[username]['restricted']}")
29
30          if database[username]["role"] == "admin" and ban == "Yes":
31              log(ERR_LOG, f"Attempt to ban admin '{username}' failed. Reason: {ADMIN_BAN_ERR}")
32              return {"Error": ADMIN_BAN_ERR}
33          old_ban = database[username]["banned"]
34
35          database[username]["banned"] = True if ban == "Yes" else False
36          database[username]["inc_att"] = 0 if ban == "No" else database[username]["inc_att"]
37          database[username]["force_change_password"] = True if ban == "Yes" else database[username]["force_change_password"]
38
39
40          if old_ban != database[username]["banned"]:
41              log(INFO_LOG, f"User '{username}' ban status changed to {database[username]['banned']}")
42
43          with open(DB_FILEPATH, 'w') as f:
44              f.write(dumps(database))
45          return username
46
47  def createUser(username:str) -> dict:
48
49      if username == "":
50          log(ERR_LOG, f"Attempt to create user failed. Reason: {EMPTY_USERNAME_ERR}")
51          return {"Error": EMPTY_USERNAME_ERR}
52
53      with open(DB_FILEPATH, 'r') as f:
54          database = loads(f.read())
55
56      if username in database:
57          log(ERR_LOG, f"Attempt to create user '{username}' failed. Reason: {USERNAME_TAKEN_ERR}")
58          return {"Error": USERNAME_TAKEN_ERR}
59
60      database[username] = {"password": EMPTY_PASS, "role": "user", "restricted": False, "force_change_password": True, "banned": False, "inc_att": 0}
61      log(INFO_LOG, f"User '{username}' created")
62      with open(DB_FILEPATH, 'w') as f:
63          f.write(dumps(database))
64
65      return {"username": username, "role": database[username]["role"]}
66
67  def deleteUser(username:str, password:str, adminname:str) -> dict:
68      if checkPassword(adminname, password):
69          with open(DB_FILEPATH, 'r') as f:
70              database = loads(f.read())
71
72          if username not in database:
73              log(ERR_LOG, f"Attempt to delete user '{username}' failed. Reason: {INC_USER_ERR}")
74              return {"Error": INC_USER_ERR}
75
76          if database[username]["role"] == "admin":
77              log(ERR_LOG, f"Attempt to delete user '{username}' failed. Reason: {ADMIN_DEL_ERR}")
78              return {"Error": ADMIN_DEL_ERR}
79
80          del database[username]
81          log(INFO_LOG, f"User '{username}' deleted")
82          with open(DB_FILEPATH, 'w') as f:
83              f.write(dumps(database))
84          return {"username": username}
85      else:
86          log(ERR_LOG, f"Attempt to delete user '{username}' failed. Reason: {INC_PASS_ERR}")
87          return {"Error": INC_PASS_ERR}

```