

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

імені ІГОРЯ СІКОРСЬКОГО”

Факультет прикладної математики

Кафедра програмного забезпечення комп’ютерних систем

**Лабораторна робота №** **2**

з дисципліни “Бази даних. Частина 2”

|  |  |  |  |
| --- | --- | --- | --- |
| Виконав  студент III курсу  групи КП-82  Мельничук Олексій Геннадійович  (*прізвище, ім’я, по батькові*)  варіант № 12 |  |  | Зарахована  “\_\_\_\_” “\_\_\_\_\_\_\_\_\_\_\_\_” 20\_\_\_ р.  викладачем  Петрашенко Андрій Васильович (*прізвище, ім’я, по батькові*) |

Київ 2021

**Мета роботи**

Здобуття практичних навичок створення ефективних програм, орієнтованих на використання сервера Redis за допомогою мови Python.

**Завдання**

Реалізувати можливості обміну повідомленнями між користувачами у оффлайн та онлайн режамах із можливістю фільтрації спам-повідомлень.

1. Проаналізувавши матеріали ресурсів, наведений у пункті “Джерела”, обрати та обгрунтувати вибір структур даних Redis щодо реалізації наведених вище вимог, обов’язково використати наступні структури даних та інструменти Redis: List, Hash, Sorted List, Set, Pub/Sub.
2. Забезпечити роботу програмних засобів у режимі емуляції із можливістю генерації повідомлень від різних користувачів, налаштування кількості виконувачів та часу затримки обробки на спам з можливістю підключення адміністратора для перегляду подій, що відбуваються.
3. Перевірку на спам можна проемулювати за допомогою затримки на псевдовипадковий час та генерацію псевдовипадкового результату (Так/Ні).

Окремі програмні компоненти та вимоги до них:

1. Redis server (RS), що виконує наступні ролі:
   1. Сховище, що містить: дані користувачів, їхні групи (звичайний користувач та адміністратор), а також повідомлення, що пересилаються між ними.
   2. Черга повідомлень, які підлягають перевірці на спам та відправленню адресату.
   3. Інструмент Publish/Subscribe для ведення та розсилання журналу активності користувачів (див. Список активностей для журналювання).
2. Інтерфейс користувача (User Interface)
   1. Звичайний користувач має змогу виконувати вхід за ім’ям (без паролю), відправляти та отримувати (переглядати) повідомлення, отримувати дані про кількість своїх повідомлень, згрупованих за статусом (див. Статуси повідомлень).
   2. Адміністратор має змогу переглядати журнал подій, що відбулись (див. Список активностей для журналювання), переглядати список користувачів, які знаходяться online, переглядати статистику (N найбільш активних відправників повідомлень із відповідною кількістю, N найактивніших “спамерів” із відповідною кількістю)
3. Виконувач (worker) призначений для:

перегляду черги повідомлень, відбору повідомлення, перевірки його вмісту на наявність спаму (у випадку наявності спаму -- додавання запису в журнал)

Інші вимоги

1. Проаналізувавши матеріали ресурсів, наведений у пункті “Джерела”, обрати та обгрунтувати вибір структур даних Redis щодо реалізації наведених вище вимог, обов’язково використати наступні структури даних та інструменти Redis: List, Hash, Sorted List, Set, Pub/Sub.
2. Забезпечити роботу програмних засобів у режимі емуляції із можливістю генерації повідомлень від різних користувачів, налаштування кількості виконувачів та часу затримки обробки на спам з можливістю підключення адміністратора для перегляду подій, що відбуваються
3. Перевірку на спам можна проемулювати за допомогою затримки на псевдовипадковий час та генерацію псевдовипадкового результату (Так/Ні). **Реалізація**

В даній роботі були застосовані такі структури даних Redis:

* List – використовується для впорядкованого збереження черги повідомлень
* Hash – використовується для збереження повідомлень та користувачів з допомогою пар ключ-значення
* Sorted list – відсортований список-«рейтинг» спамерів та відправників
* Set – список користувачів онлайн з униканням дублювання (всі значення унікальні)
* Pub/sub – використовується для журналу подій

Програма має дані скрипти:

* main.py – консольний інтерфейс для користувача та адміністратора; є можливість запустити декілька вікон консолі з різними користувачами.
* emulation.py – емуляція що тестує програму, створюючи нових юзерів та надсилаючи повідомлення
* worker.py – переглядувач черги повідомлень, що відокремлює спам

В папці ops:

* connection.py – конфігурація для підключення до бд Redis
* message.py – файл з класом Message що описує всі операції з об’єктом
* user.py – файл з класом User що описує всі операції з об’єктом

**Код програми**

|  |
| --- |
| **Main.py – консольний інтерфейс** |
| import os  import logging  from pathlib import Path  from ops.user import User  from ops.message import Message  import ops.connection as redis  redis.connect()  rconnection = redis.connection  import re  FILE = os.path.join(os.path.dirname(Path(\_\_file\_\_).absolute()), 'logs.txt')  logging.basicConfig(filename="logs.txt", level=logging.INFO)  def start\_menu():  print("\nMAIN MENU")  print("1. Register")  print("2. Login")  print("0. Exit")  return int(input("Enter the number of action: "))  def user\_menu(uid):  print(f"\nUSER MENU: {User.get\_username(uid)}")  print("1. Inbox")  print("2. Write new message")  print("3. Message status")  print("4. Give admin")  print("0. Logout")  return int(input("Enter the number of action: "))  def main():  while True:  action = start\_menu()  #register  if action == 1:  username = input("Enter new username: ")  uid = User.register(username)    if User.is\_logged\_in(uid):  user(uid)  #login  elif action == 2:  username = input("Enter username: ")  uid = User.login(username)    if User.is\_logged\_in(uid):  user(uid)  #exit  elif action == 0:  print("Farewell!")  break  #wrong choice  else:  print("Enter correct choice (num 0 to 2): ")  def admin\_menu(uid):  print(f"\nADMIN MENU: {User.get\_username(uid)}")  print("1. Sender rating")  print("2. Spammer rating")  print("3. View logs")  print("4. Users online")  print("0. Exit")  return int(input("Enter the number of action: "))  def user(uid):  while True:  action = user\_menu(uid)  #inbox  if action == 1:  Message.get\_inbox(uid)  #new message  elif action == 2:  text = input("Enter message: ")  receiver = input("Enter username of the receiver: ")    if Message.create\_message(uid, text, receiver):  print(f"Sent message to {receiver}!")  else:  print("Got some trouble with sending a message")  #message status  elif action == 3:  keys = ["queue", "checking", "blocked", "sent", "delivered"]  view = ["Queued", "Checking", "Blocked", "Sent", "Delivered"]  user = rconnection.hmget(f"user{uid}", keys)    for i in range(5):  print(f"-{view[i]}: {user[i]}")    #get admin  elif action == 4:  return admin(uid)  #logout  elif action == 0:  User.logout(uid)  break  #incorrect  else:  print("Enter correct choice (num 0 to 4): ")  def admin(uid):  while True:  action = admin\_menu(uid)  #get senders  if action == 1:  quantity = 10  active\_senders = rconnection.zrange("sent", 0, quantity, desc=True, withscores=True)  if len(active\_senders) == 0:  print("No senders found.")  else:  print(f"All {len(active\_senders)} most active senders: ")  for index, sender in enumerate(active\_senders):  sid = re.search(r'\d+', sender[0]).group()  uname = User.get\_username(sid)  print(f"{index + 1}. {uname} - {int(sender[1])} messages")  #get spammers  elif action == 2:  quantity = 10  active\_spamers = rconnection.zrange("spam", 0, quantity, desc=True, withscores=True)  if len(active\_spamers) == 0:  print("No spamers found.")  else:  print(f"All {len(active\_spamers)} most active spamers: ")  for index, sender in enumerate(active\_spamers):  sid = re.search(r'\d+', sender[0]).group()  uname = User.get\_username(sid)  print(f"{index + 1}. {uname} - {int(sender[1])} messages")  #get logs  elif action == 3:  try:  with open(FILE) as file:  print(file.read())  except Exception as e:  return f"Error: Log reading: '{e}'"  #online members  elif action == 4:  online\_users = rconnection.smembers("online")  if len(online\_users) == 0:  print("No one's around.")  else:  print(f"{len(online\_users)} users online: ")  for user in online\_users:  print(f"-{user}")  #logout  elif action == 0:  print(f"Farewell, {User.get\_username(uid)}!")  break  #incorrect  else:  print("Enter correct choice (num 0 to 4): ")  if \_\_name\_\_ == '\_\_main\_\_':  main() |

|  |
| --- |
| **emulation.py – генерування даних** |
| import ops.connection as redis  redis.connect()  rconnection = redis.connection  from ops.user import User  from ops.message import Message  from threading import Thread  from faker import Faker  from random import randint  import atexit  class Emulation(Thread):  def \_\_init\_\_(self, name, users):  Thread.\_\_init\_\_(self)  self.conn = rconnection  self.name = name  self.users = users  self.user\_id = User.register(name)  def run(self):  for i in range(amount):  sentence = fake.sentence(nb\_words=5, variable\_nb\_words=True, ext\_word\_list=None)  receiver = users[randint(0, amount - 1)]  print(f"Message {sentence} was sent to {receiver}");  Message.create\_message(self.user\_id, sentence, receiver)  def exit():  online = rconnection.smembers("online")  for i in online:  rconnection.srem("online", i)  rconnection.publish("logout", f"User {i} signed out.")  print(f"{i} exits app. Have a good day!")  if \_\_name\_\_ == '\_\_main\_\_':  fake = Faker()  atexit.register(exit)  amount = 5    users = [fake.profile(fields=["username"], sex=None)["username"] for user in range(amount)]  threads = []  for i in range(amount):  print(f"User: {users[i]}")  threads.append(Emulation(users[i], users))  for t in threads:  t.start() |

|  |
| --- |
| **Worker.py – переглядувач черги + перевірка на спам** |
| import ops.connection as redis  redis.connect()  rconnection = redis.connection  from ops.message import Message  from ops.user import User  from threading import Thread  from random import randint  import time  import random  import logging  import datetime  logging.basicConfig(filename="logs.txt", level=logging.INFO)  def is\_spam():  return random.random() > 0.5  class Worker(Thread):  def \_\_init\_\_(self):  Thread.\_\_init\_\_(self)  def run(self):  message = rconnection.brpop("queue")  if message:  message\_id = message[1]  message\_key = f"message{message\_id}"  rconnection.hset(message\_key, "status", "checking")  message = rconnection.hmget(message\_key, ["sender\_id", "receiver\_id"])  sender\_id = message[0]  receiver\_id = message[1]  sender\_name = User.get\_username(sender\_id)  rconnection.hincrby(f"user{sender\_id}", "queue", -1)  print("Message enqueued")  rconnection.hincrby(f"user{sender\_id}", "checking", 1)  time.sleep(randint(0, 3))  pipeline = rconnection.pipeline(True)  pipeline.hincrby(f"user{sender\_id}", "checking", -1)  if is\_spam():  print(f"{sender\_name} sent spam: id={message\_id}")  message\_text = rconnection.hmget(message\_key, ["text"])[0]  logging.info(f"({datetime.datetime.now()}): User {sender\_name} sent spam: {message\_text}")  pipeline.zincrby(f"spam", 1, f"user{sender\_id}")  pipeline.hset(message\_key, "status", "blocked")  pipeline.hincrby(f"user{sender\_id}", "blocked", 1)  pipeline.publish("spam", f"User {sender\_name} sent spam: {message\_text}.")  else:  print(f"Checked and sent message[{message\_id}] from {sender\_name}.")  pipeline.hset(message\_key, "status", "sent")  pipeline.hincrby(f"user{sender\_id}", "sent", 1)  pipeline.sadd(f"sent\_to:{receiver\_id}", message\_id)    pipeline.execute()  def main():  handlers = 5  for i in range(handlers):  worker = Worker()  worker.daemon = True  worker.start()  while True:  pass  if \_\_name\_\_ == '\_\_main\_\_':  main() |

|  |
| --- |
| **ops/User.py – операції з користувачами** |
| import ops.connection as redis  import logging  import datetime  logging.basicConfig(filename="logs.txt", level=logging.INFO)  redis.connect()  rconnection = redis.connection  class User:  def register(username):  if rconnection.hget("users", username):  print(f"{username} already exists.")  return -1  user\_id = rconnection.incr("user\_id")  user\_key = f"user{user\_id}"  user\_info = {  "id": user\_id,  "name": username,  "queue": 0,  "checking": 0,  "blocked": 0,  "sent": 0,  "delivered": 0  }  rconnection.hset("users", username, user\_id)  for key in user\_info.keys():  rconnection.hset(user\_key, key, user\_info[key])  rconnection.publish("register", f"User {username} registered")  rconnection.sadd("online", username)  logging.info(f"({datetime.datetime.now()}): User {username} registered")  return user\_id  def login(username):  user\_id = rconnection.hget("users", username)  if not user\_id:  print(f"{username} does not exist. Register?")  return -1  rconnection.publish("login", f"User {username} logged in")  rconnection.sadd("online", username)  logging.info(f"({datetime.datetime.now()}): User {username} logged in")  return user\_id  def logout(user\_id):  username = User.get\_username(user\_id)  rconnection.publish("logout", f"User {username} logged out")  rconnection.srem("online", username)  logging.info(f"({datetime.datetime.now()}): User {username} logged out")  def get\_username(user\_id):  return rconnection.hmget(f"user{user\_id}", ["name"])[0]  def is\_logged\_in(user\_id):  return user\_id != -1 |

|  |
| --- |
| **ops/Message.py – операції з повідомленнями** |
| from os import pipe  import ops.connection as redis  redis.connect()  rconnection = redis.connection  from ops.user import User  import random  class Message:  def create\_message(user\_id, message, receiver):  message\_id = rconnection.incr("message\_id")  receiver\_id = rconnection.hget("users", receiver)  if not receiver\_id:  print(f"{receiver} does not exist, can't send a message.")  return False  message\_key = f"message{message\_id}"  message\_info = {  "id": message\_id,  "text": message,  "sender\_id": user\_id,  "receiver\_id": receiver\_id,  "status": "created"  }  pipeline = rconnection.pipeline(True)  for key in message\_info.keys():  pipeline.hset(message\_key, key, message\_info[key])  pipeline.lpush("queue", message\_id)  pipeline.hset(message\_key, "status", "queue")  pipeline.hincrby(f"user{user\_id}", "queue", 1)  pipeline.zincrby("sent", 1, f"user{user\_id}")  pipeline.execute()  return message\_id  def get\_inbox(user\_id):  messages = rconnection.smembers(f"sent\_to{user\_id}")  if len(messages) == 0:  print("No messages")  return  for message\_id in messages:  message = rconnection.hmget(f"message{message\_id}", ["text", "status", "sender\_id"])  if message[1] != "delivered":  rconnection.hset(f"message{message\_id}", "status", "delivered")  rconnection.hincrby(f"user{message[2]}", "sent", -1)  rconnection.hincrby(f"user{message[2]}", "delivered", 1)  print(f"{message[0]} -> FROM: {User.get\_username(message[2])}") |

|  |
| --- |
| **ops/connection.py – конфіг підключення** |
| import redis  import sys  class Redis:  host = 'localhost'  port = 6379  connection = redis.Redis(host=Redis.host, port=Redis.port, db=0, decode\_responses=True)  def connect():  try:  connection.ping()  except Exception as err:  sys.exit(err) |

**Результати роботи програм**

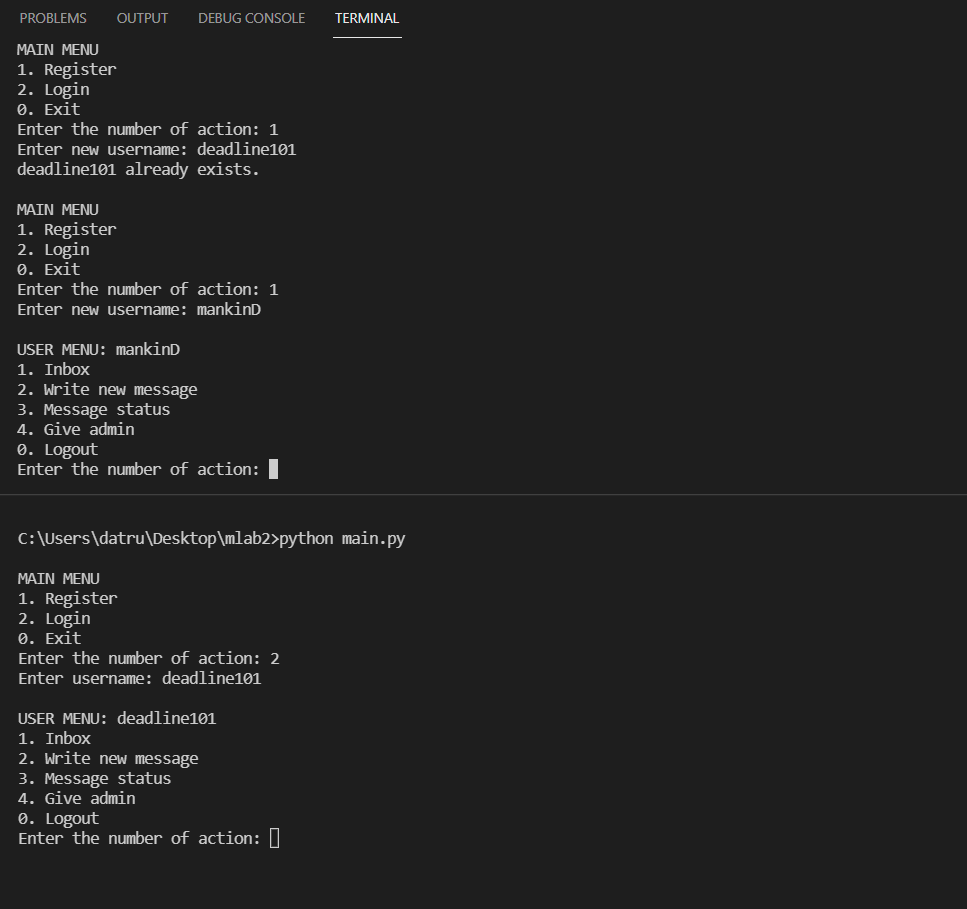
****

Рис. 1. Два користувачі відкриті в двох консолях водночас.

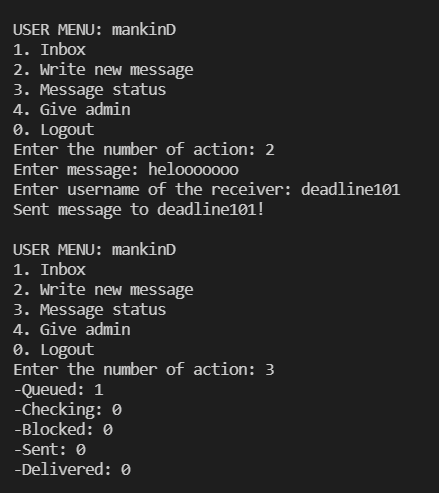
****

Рис. 2. Надсилання повідомлень.

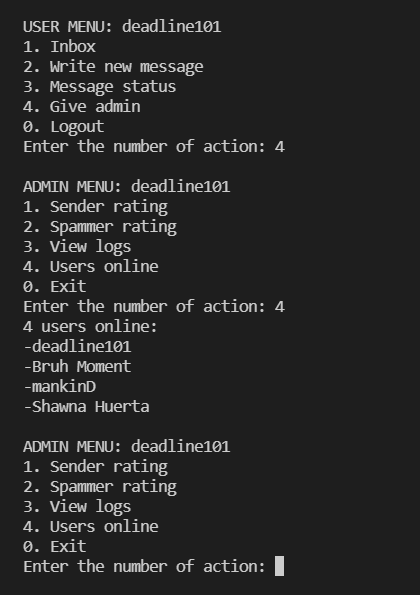


Рис. 3. Функція “Give admin” надає права адміністратора користувачу та надає доступ до нових функцій, таких як перегляд користувачів онлайн.

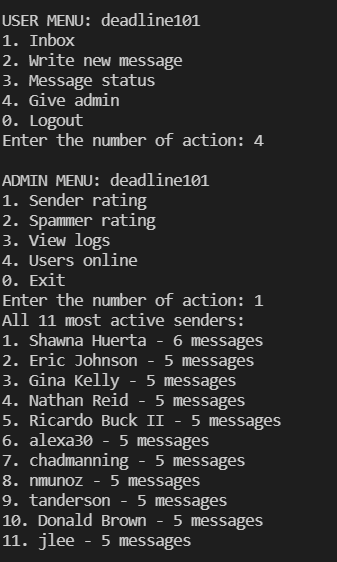
****

Рис. 4. Перегляд найактивніших відправників.

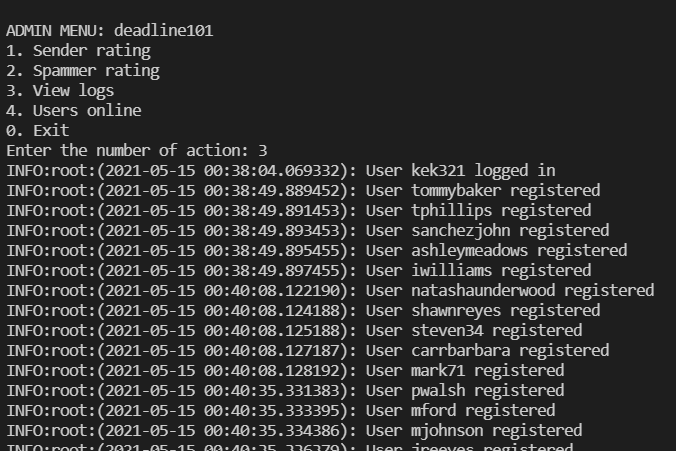


Рис. 5. Перегляд логів

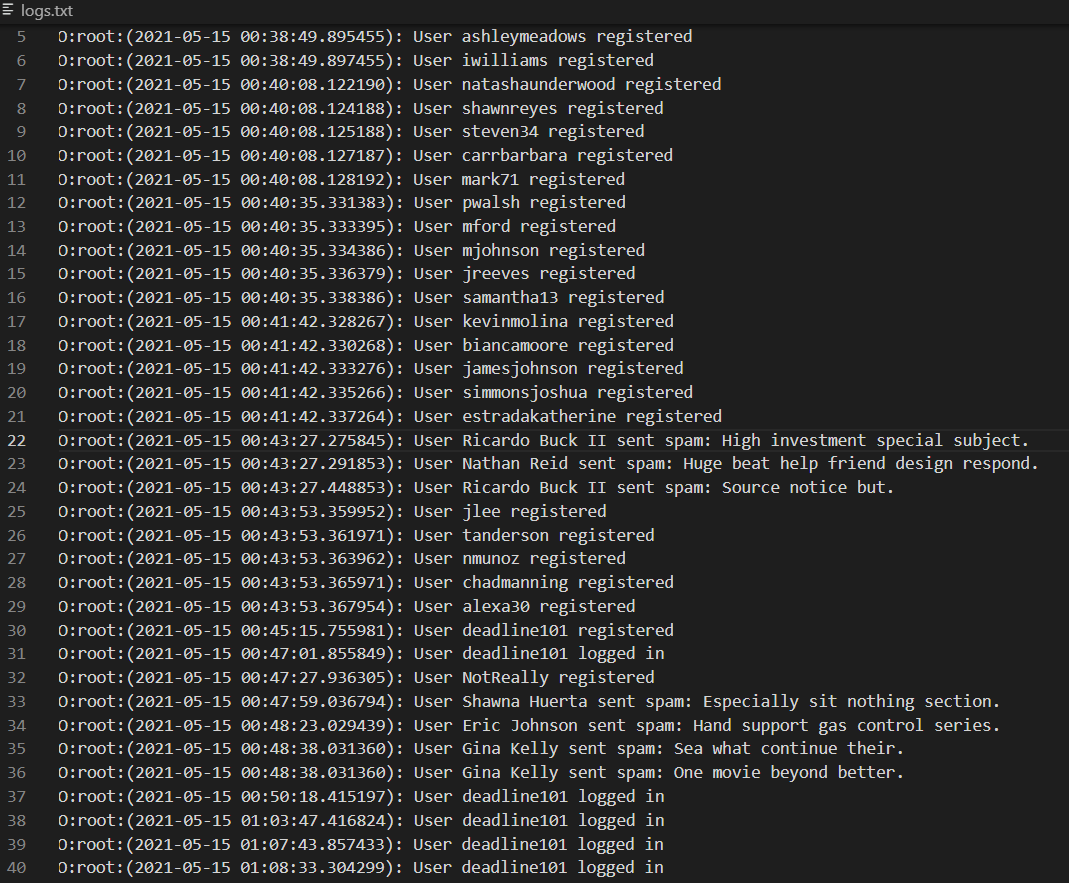
****

Рис. 6. Логування **Відповіді на контрольні запитання**

1. Визначити сфери застосування основних структури даних redis (List, Hash, Sorted List, Set, Pub/Sub).

List – зручні для черг та стеків для швидкого доступу до верхньго/нижнього елементів

Hash – зручна структура для об’єктів та структур з полями з різними значеннями

Sorted List – реалізація «топів» та «рейтингів» з множин за різними параметрами

Set – множини в яких можна швидко перевірити наявність елемента в колекції

Pub/Sub – легковагова розсилка повідомлень, зручна для швидких малих повідомлень

1. Визначити основні переваги та недоліки redis.

Переваги:

* Висока швидкість
* Простий в налаштуванні та установці
* Зручність в роботі з різними типами даних
* Гарна документація та підтримка
* Open source

Недоліки

* Потреба великої кількості оперативної пам’яті, оскільки всі дані знаходяться там
* Не має складної мови запитів, лише команди
* Масштабування складне й частіше за все дороге у налаштуванні
* Все навантаження зосереджується на одному master