МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Компьютерные науки и прикладная математика»

Кафедра №806 «Вычислительная математика и программирование»

**Лабораторная работа 2**

**по курсу «Базы Данных»**

Выполнил: Куценко М.Д.

Группа: М8О-312Б-22

Преподаватель: А.В. Малахов

Москва, 2025

**Оглавление**

[2.0 Redis — конфигурация 2](#__RefHeading___Toc269_596683903)

[2.1 Хранение даннных 2](#__RefHeading___Toc269_5966839031)

[2.2 PubSub 3](#__RefHeading___Toc269_5966839032)

[2.3 Реализация в приложении 3](#__RefHeading___Toc1666_2656672115)

[Вывод 4](#__RefHeading___Toc321_596683903)

# **2.0 Redis — конфигурация**

Создаём Redis в Docker-контейнере через docker-compose:

|  |
| --- |
| redis: image: redis:latest container\_name: redis ports: - "6379:6379" |

Подключаемся к контейнеру через библиотеку redis-py:

|  |
| --- |
| from redis import Redis  redis = Redis() |

# **2.1 Хранение даннных**

Код для записи, чтения, проверки существования записи и удаления записи:

|  |
| --- |
| def set(key: str, value: str, ttl: int = None) -> bool:   actual\_ttl = ttl if ttl is not None else int(os.getenv("EXPIRE\_TIME", "3600"))      result = redis.set(key, value)      if actual\_ttl > 0:       redis.expire(key, actual\_ttl)      return result    def get(key: str):   data = redis.get(key)   return data  def delete(key: str) -> bool:   return bool(redis.delete(key))  def exists(key: str) -> bool:   return bool(redis.exists(key)) |

# **2.2 PubSub**

Код для чтения из топика:

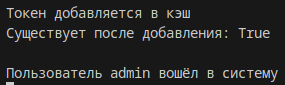
|  |
| --- |
| def redis\_listener(channels: list):    pubsub = redis.pubsub()    pubsub.subscribe(channels)    while(True):        message = pubsub.get\_message(ignore\_subscribe\_messages=True)       if message != None:           print(message['data'].decode('utf-8'))        time.sleep(0.5) |

# **2.3 Реализация в приложении**

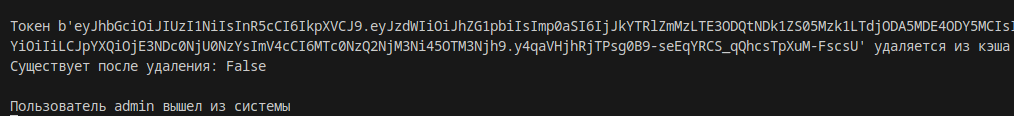
Создание Redis в session\_state у Streamlit, создание потока который в фоновом процессе читает сообщения через pusub:

|  |
| --- |
| if 'redis' not in st.session\_state:   st.session\_state.redis = Redis()  if 'listener\_started' not in st.session\_state:   thread = threading.Thread(target=redis\_listener, args=[[channel\_name1, channel\_name2]], daemon=True)   thread.start()   st.session\_state.listener\_started = True  def publish\_channel(channel\_name, message):   st.session\_state.redis.publish(channel\_name, message)  def add\_token():   print("\nТокен добавляется в кэш")   set(st.session\_state.username, make\_token(st.session\_state.username))   print(f"Существует после удаления: {exists(st.session\_state.username)}\n")  def delete\_token():   print(f"\nТокен {get(st.session\_state.username)} удаляется из кэша")   delete(st.session\_state.username)   print(f"Существует после удаления: {exists(st.session\_state.username)}\n") |

Пользователь входит в систему:



Пользователь выходит из системы:

 Другой пользователь видит действия текущего через pubsub:



# **Вывод**

В ходе выполнения лабораторной работы была проведена оптимизация работы курсового проекта за счёт внедрения Redis для работы с динамическими данными. Для JWT-токенов авторизации было реализовано хранение в Redis с настройкой TTL, что обеспечивает автоматическое удаление просроченных токенов и снижает нагрузку на основную базу данных. Настройка мгновенных оповещений через PubSub позволила оперативно уведомлять клиентов о входе других пользователей.

Redis снижает нагрузку на основную базу данных, уменьшает время отклика за счёт хранения данных в оперативной памяти и предоставляет эффективные механизмы для работы с динамическими данными. В результате система стала более отзывчивой, масштабируемой и устойчивой к пиковым нагрузкам.