**Интеграция Reddis в проект**

**Redis**— это быстрая база данных вида ключ-значение, которая хранит данные в оперативной памяти (RAM), а не на жестком диске.

**Как кэш**: Ускоряет работу сайтов и приложений, сохраняя часто используемые данные (например, результаты запросов) для мгновенного доступа.

**Как хранилище**: Может работать с разными типами данных — строки, списки, хэши

**Очень быстрый:** Так как данные в памяти, ответы приходят за доли миллисекунд.

Подходит для очередей, чатов, счетчиков, сессий пользователей и многого другого.

1. **Добавим Редис как сервис для сборки проекта. Выделим ему том и замаппим к порту.**

redis:

    container\_name: redis

    image: redis:7-alpine

    ports:

      - "6379:6379"

    volumes:

      - redis\_data:/data

1. **Напишем клиентские операции с редисом.**

*from* redis *import* asyncio *as* aioredis

*import* json

*from* src.config *import* settings

*# Создаём соединение*

redis = aioredis.from\_url(settings.REDIS\_URL, *decode\_responses*=True)

#Положить токен в редис кэш

async def **store\_token**(*user\_id*: str, *token*: str):

*await* redis.setex(

        f"token:{*user\_id*}",

        settings.TOKEN\_EXPIRE\_DAYS \* 24 \* 60 \* 60,  *#Перевести в секунды*

*token*

    )

#Получить токен из редис кэша

async def **get\_redis\_token**(*user\_id*: str) -> str:

*return* *await* redis.get(f"token:{*user\_id*}")

#Удалить токен при ручном выходе

async def **invalidate\_token**(*user\_id*: str):

*await* redis.delete(f"token:{*user\_id*}")

*# Кэшируем, например, данные о питомце*

async def **cache\_pet\_data**(*user\_id*: int, *pets\_data*: list):

    """Cache pet data in Redis using hash"""

    key = f"pets:{*user\_id*}"

*await* redis.setex(

        key,

        300,  *# Храним 5 минут*

        json.dumps(*pets\_data*)

    )

#Достаем данные о питомцах пользователя с user\_id

async def **get\_cached\_pet\_data**(*user\_id*: int) -> list:

    """Get cached pet data from Redis"""

    key = f"pets:{*user\_id*}"

    data = *await* redis.get(key)

*return* json.loads(data) *if* data *else* None

#Функция для верификации токена

async def **verify\_token**(*token*: str, *user\_id*: str) -> bool:

    """Verify if token exists in Redis and matches"""

    stored\_token = *await* get\_token(*user\_id*)

*return* stored\_token == *token*

*# PubSub*

async def **publish\_event**(*channel*: str, *message*: dict):

    """Publish event to Redis channel"""

*await* redis.publish(*channel*, json.dumps(*message*))

async def **subscribe\_to\_events**(*channels*: list):

    """Subscribe to Redis channels"""

    pubsub = redis.pubsub()

*await* pubsub.subscribe(\**channels*)

*return* pubsub

1. **Будем добавлять токен в кэш при его создании**

async def **create\_access\_token**(*data*: dict) -> str:

    to\_encode = *data*.copy()

    expire = datetime.now(timezone.utc) + timedelta(*days*=30)

    to\_encode.update({"exp": expire})

    auth\_data = get\_auth\_data()

    encode\_jwt = jwt.encode(to\_encode, auth\_data['secret\_key'], *algorithm*=auth\_data['algorithm'])

*# Добавляем в Редис токен*

*await* store\_token(str(*data*['sub']), encode\_jwt)

*return* encode\_jwt

1. **Будем удалять токен из кэша при выходе из аккаунта**

@router.post("/logout/")

async def **logout\_user**(*response*: Response, *user*: SUserGet = Depends(get\_current\_user)):

*# Удаляем токен из кэша*

*await* invalidate\_token(str(*user*.user\_id))

*response*.delete\_cookie(*key*="users\_access\_token")

*return* {'message': 'Пользователь успешно вышел из системы'}

1. **Реализуем получение информации о питомцах человека через кеш редис**

async def **get\_current\_pets**(*user*: SUserGet = Depends(get\_current\_user)) -> List[SPetGet]:

*# Птыаемся сначала получить JSON из редис кеша*

    cached\_pets = *await* get\_cached\_pet\_data(*user*.user\_id)

*if* cached\_pets:

*return* [SPetGet(\*\*pet) *for* pet *in* cached\_pets]

*# Если не в кэше – смотрим в базу данных*

*async* *with* async\_session\_factory() *as* session:

*async* *with* session.begin():

            pets\_query = text("""

                SELECT pet\_id, name, sex, animal, breed, birth, weight, sterilized

                FROM pets

                WHERE owner\_id = :user\_id

            """)

            result = *await* session.execute(pets\_query, {"user\_id": *user*.user\_id})

            pets = result.fetchall()

--------------------------------------------

--------------------------------------------

*# Получив данные из ДБ, кладем их в редис*

*await* cache\_pet\_data(*user*.user\_id, [pet.model\_dump() *for* pet *in* pet\_list])

*return* pet\_list

1. **Реализуем проверку токена через кеш редис**

async def **get\_current\_user**(*token*: str = Depends(get\_token)):

*try*:

*# Верификация токена, сравнение с токеном в кеше*

    stored\_token = *await* get\_redis\_token(user\_id)

*if* not stored\_token or stored\_token != *token*:

*raise* HTTPException(*status\_code*=status.HTTP\_401\_UNAUTHORIZED, *detail*='Token is invalid or expired')

*async* *with* async\_session\_factory() *as* session:

*async* *with* session.begin():

            user\_query = text("SELECT \* FROM users WHERE user\_id = :user\_id")

            result = *await* session.execute(user\_query, {"user\_id": int(user\_id)})

**PubSub**

**PubSub**— это способ обмена сообщениями между разными частями программы или между серверами.

Издатель (Publisher) — отправляет сообщения в канал (как телевизионный канал).

Подписчик (Subscriber) — слушает этот канал и получает сообщения, как только они появляются.

**Отправляем сообщения при успешной аутентификации пользователя**

@router.post("/login")

async def **auth\_user**(*response*: Response,

*user\_data*: SUserAuth):

    check = *await* authenticate\_user(*mail* = *user\_data*.mail, *password*= *user\_data*.password)

*if* check is None:

*raise* HTTPException(*status\_code*=status.HTTP\_401\_UNAUTHORIZED, *detail*= "Неверная почта или пароль")

    access\_token = *await* create\_access\_token({"sub": str(check)})

*response*.set\_cookie(*key* = "users\_access\_token", *value*= access\_token, *httponly*= True)

*# Отправляем сообщение*

*await* publish\_event("user\_events", {

        "type": "login",

        "user\_id": check,

        "timestamp": datetime.now().isoformat()

    })

*return* {'access\_token': access\_token, "refresh\_token": None}

**Отправляем сообщение при добавлении нового питомца**

@router.post("/my\_pets/add\_pets")

async def **add\_pet**(

*pet\_data*: SPetAdd,

*disease\_data*: Optional[List[SDiseaseAdd]] = None,

*feed\_data*: Optional[SFeedAdd] = None,

*user*: SUserGet = Depends(get\_current\_user)

):

    pet\_id = *await* add\_pet\_to\_db(*user*.user\_id, *pet\_data*, *disease\_data*, *feed\_data*)

*# Отправляем сообщение*

*await* publish\_event("pet\_events", {

        "type": "new\_pet",

        "user\_id": *user*.user\_id,

        "pet\_id": pet\_id,

        "pet\_name": *pet\_data*.name,

        "timestamp": datetime.now().isoformat()

    })

*return* {"message": "Питомец успешно добавлен", "pet\_id": pet\_id}

**Отправляем сообщение при удалении питомца**

@router.delete("/my\_pets/{pet\_id}")

async def **delete\_pet**(*pet\_id*: int, *user*: SUserGet = Depends(get\_current\_user)):

*async* *with* async\_session\_factory() *as* session:

*async* *with* session.begin():

            delete\_diseases\_query = text("DELETE FROM pets\_diseases WHERE pet\_id = :pet\_id")

*await* session.execute(delete\_diseases\_query, {"pet\_id": *pet\_id*})

*# Сообщение об удалении*

*await* publish\_event("pet\_events", {

        "type": "pet\_deleted",

        "user\_id": *user*.user\_id,

        "pet\_id": *pet\_id*,

        "timestamp": datetime.now().isoformat()

    })

*return* {"message": "Питомец и его связанные записи успешно удалены"}

**Результат**

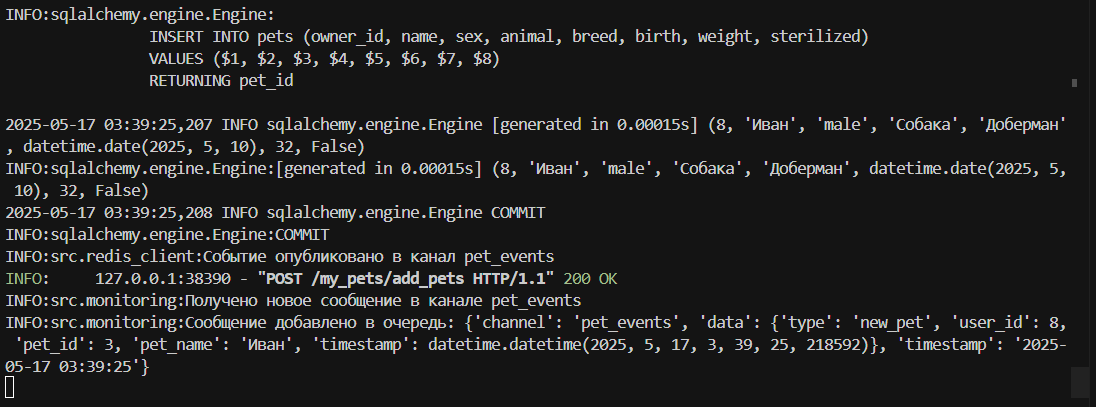
Здесь представлено время выполнения запроса списка питомцев для пользователя.

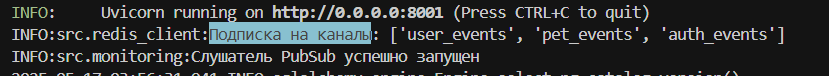
Первый запрос был отправлен к базе данных и занял 15 миллисекунд. Ответ был закеширован. Далее, не делая новых записей, были отправлены последующие запросы, которые занимали примерно по 4 миллисекунды, потому что шли не к базе данных, а к кешу Редис.

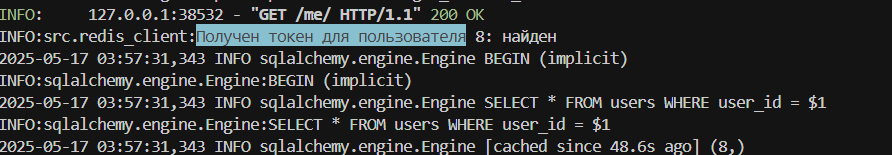
****

**Отслеживание событий**

В терминале можно видеть сообщения от издателя о том, что был добавлен новый питомец в базу данных для пользователя с user\_id = 8

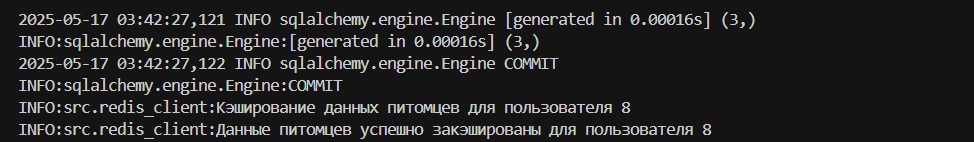


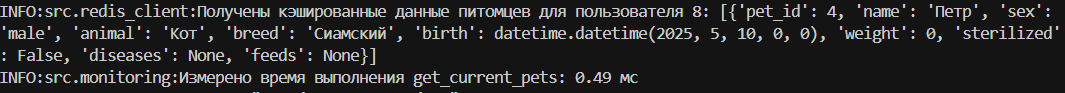
В интерфейсе так же отображается список последних событий с меткой времени и канала.

Аналогично отображается, найден ли в кеше токен или нет

Здесь для примера сообщение из канала pet\_events о добавлении и удалении нового питомца и сообщение из канала user\_events о входе пользователя с user\_id = 8



Здесь можно видеть сообщение от Redis о том, что данные о питомцах успешно закешировались

Здесь сообщение о том, что при очередном запросе питомцев для пользователя данные вернулись из кеша Redis