Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
УРАЛЬСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ
ИМЕНИ ПЕРВОГО ПРЕЗИДЕНТА РОССИИ Б.Н. ЕЛЬЦИНА
(УрФУ имени первого Президента России Б.Н. Ельцина)
Институт радиоэлектроники и информационных технологий — РТФ
Школа профессионального и академического образования

ОСНОВЫ РАБОТЫ С REDIS

Отчет по лабораторной работе №7

по дисциплине «Разработка приложений»

	Дата	Подпись	
Преподаватель:	_____	_____	Кузьмин Д.И.
Студенты:	_____	_____	Пантелеев Е.А.

Группа: РИМ-150950

Екатеринбург

2025

ЦЕЛЬ РАБОТЫ

Овладеть базовыми навыками установки, подключения и взаимодействия с Redis в Python. Изучить основные структуры данных Redis и их применение на практике. основные принципы работы с брокером сообщений RabbitMQ в Python, изучить различные паттерны обмена сообщениями и научиться создавать распределенные приложения.

ХОД РАБОТЫ

Обновляем файл docker compose с redis.

```
32
    ▷Run Service
33  redis:
34      image: redis:latest
35      container_name: redis
36      ports:
37        - "6379:6379"
38      volumes:
39        - redis-data:/data
40
41  volumes:
42    redis-data:
```

Сделаем тестовый скрипт test_redis.py для проверки работы redis.

```
test_redis.py > ...
1   from app.cache.redis_client import get_redis
2
3   print("Тестируем Redis модуль...")
4
5   redis_client = get_redis()
6
7   if redis_client:
8       print("✅ Redis подключен")
9
10      # Тест записи
11      redis_client.set("project:test", "lab6")
12      value = redis_client.get("project:test")
13      print(f"✅ Прочитали: {value}")
14
15      # Очистка
16      redis_client.delete("project:test")
17  else:
18      print("❌ Redis недоступен")
19
```

Файл redis_client.py выглядит следующим образом. Здесь происходит только инициализация подключения к контейнеру redis.

app > cache > redis_client.py > get_redis

```python
import redis


GigaCode: explain | explain step by step | doc | test
def get_redis():
    """Возвращает подключение к Redis или None"""
    try:
        client = redis.Redis(
            host="redis",
            port=6379,
            db=0,
            decode_responses=True,
            socket_connect_timeout=2,
        )
        client.ping()
        return client
    except (redis.ConnectionError, redis.TimeoutError):
        return None

```

Предварительно поставим host=localhost для подключения в контейнер и проверим подключение.

```
(.venv) C:\Users\highs\OneDrive\Документы\GitHub\lab6>python test_redis.py
Тестируем Redis модуль...
✅ Redis подключен
✅ Прочитали: lab6
```

Для добавления кэширования юзеров добавим кэширование с подробным логированием в user_service.py

```python
    async def get_by_id(self, session: Session, user_id: int) -> Optional[User]:
        """Получить пользователя по ID с кешированием"""
        redis_client = get_redis()

        # Пробуем получить из кеша
        if redis_client:
            cached_data = redis_client.get(self._cache_key(user_id))
            if cached_data:
                logger.info(f"🟢 [CACHE HIT] User {user_id} found in Redis")
                try:
                    data = json.loads(cached_data)
                    return self._dict_to_user(data)
                except (json.JSONDecodeError, KeyError) as e:
                    logger.warning(
                        f"🔴 [CACHE ERROR] Invalid cache for user {user_id}: {e}"
                    )
                    # Удаляем повреждённый кеш
                    try:
                        redis_client.delete(self._cache_key(user_id))
                    except (json.JSONDecodeError, ValueError):
                        pass
            else:
                logger.info(f"🟡 [CACHE MISS] User {user_id} not in Redis")
        else:
            logger.info(f"🔵 [NO REDIS] Redis not available for user {user_id}")

        # Получаем из БД
        logger.info(f"📊 [DB QUERY] Fetching user {user_id} from database")
        user = await self.user_repository.get_by_id(session, user_id)

        # Сохраняем в кеш
        if user and redis_client:
            logger.info(f"💾 [CACHE SAVE] Saving user {user_id} to Redis")
            try:
                data = self._user_to_dict(user)
                redis_client.setex(self._cache_key(user_id), 3600, json.dumps(data))
                logger.info(f"✅ [CACHE SAVED] User {user_id} cached successfully")
            except (json.JSONDecodeError, ValueError) as e:
                logger.warning(f"❌ [CACHE FAILED] Failed to cache user {user_id}: {e}")

        return user
```

Попробуем получить данные юзера и проверить логи. Получаем юзера в первый раз, видим, что в кэше их нет, и затем кэш обновляется.

```
(.venv) C:\Users\highs\OneDrive\Документы\GitHub\lab6>curl "http://localhost:8000/users/2"
{"id":2,"username":"petrov","email":"petrov@example.com","description":"Программист и геймер","created_at":"2025-12-15T11:11:
24.350379","updated_at":"2025-12-15T11:11:25.251952"}
```

```
app-1  | INFO:     172.18.0.1:57018 - "GET /users/1 HTTP/1.1" 200 OK
app-1  | INFO - 2025-12-15 11:15:41,630 - app.services.user_service - user_service - 🟡 [CACHE MISS] User 2 n ot in Redis

app-1  | INFO - 2025-12-15 11:15:41,630 - app.services.user_service - user_service - 📊 [DB QUERY] Fetching user 2 from datab
ase
```

```
app-1  | 2025-12-15 11:15:41,638 INFO sqlalchemy.engine.Engine ROLLBACK
app-1  | INFO - 2025-12-15 11:15:41,638 - sqlalchemy.engine.Engine - base - ROLLBACK
app-1  | INFO:     172.18.0.1:45272 - "GET /users/2 HTTP/1.1" 200 OK
app-1  | INFO - 2025-12-15 11:16:35,552 - app.services.user_service - user_service - 🟢 [CACHE HIT] User 2 fo und in Redis

app-1  | INFO:     172.18.0.1:58566 - "GET /users/2 HTTP/1.1" 200 OK
```

Попробуем сделать update и посмотрим на изменение в кэше.

```
(.venv) C:\Users\highs\OneDrive\Документы\GitHub\lab6>curl -X PUT "http://localhost:8000/users/1" -H "Content-Type: applicati
on/json" -d "{\"email\":\"updated@test.ru\"}"
{"id":1,"username":"ivanov","email":"updated@test.ru","description":"Любитель путешествий и фотографии","created_at":"2025-12
-15T11:11:24.350373","updated_at":"2025-12-15T11:16:56.559178"}
```

```
app-1  | 2025-12-15 11:16:56,567 INFO sqlalchemy.engine.Engine [generated in 0.00023s] (1,)
app-1  | INFO - 2025-12-15 11:16:56,566 - sqlalchemy.engine.Engine - base - BEGIN (implicit)
app-1  | INFO - 2025-12-15 11:16:56,567 - sqlalchemy.engine.Engine - base - SELECT users.id, users.username, users.email, use
rs.description, users.created_at, users.updated_at
app-1  | FROM users
app-1  | WHERE users.id = ?
app-1  | INFO - 2025-12-15 11:16:56,567 - sqlalchemy.engine.Engine - base - [generated in 0.00023s] (1,)
app-1  | INFO - 2025-12-15 11:16:56,571 - app.services.user_service - user_service - 🗑️[CACHE INVALIDATE] D eleting cache fo
r user 1
app-1  | INFO - 2025-12-15 11:16:56,572 - app.services.user_service - user_service - ✅ [CACHE DELETED] Cache for user 1 remo
v
ed
app-1  | 2025-12-15 11:16:56,573 INFO sqlalchemy.engine.Engine ROLLBACK
app-1  | INFO - 2025-12-15 11:16:56,573 - sqlalchemy.engine.Engine - base - ROLLBACK
app-1  | INFO:     172.18.0.1:54172 - "PUT /users/1 HTTP/1.1" 200 OK
```

Теперь попробуем остановить контейнер redis и обратиться к нему.

```
(.venv) C:\Users\highs\OneDrive\Документы\GitHub\lab6>docker stop redis
redis

(.venv) C:\Users\highs\OneDrive\Документы\GitHub\lab6>curl "http://localhost:8000/users/2"
{"id":2,"username":"petrov","email":"petrov@example.com","description":"Программист и геймер","created_at":"2025-12-15T11:11:
24.350379","updated_at":"2025-12-15T11:11:25.251952"}
```

```
app-1  | 2025-12-15 11:18:05,514 INFO sqlalchemy.engine.Engine BEGIN (implicit)
app-1  | INFO - 2025-12-15 11:18:05,513 - app.services.user_service - user_service - 🔵 [NO REDIS] Redis not available for us
er 2
app-1  | INFO - 2025-12-15 11:18:05,513 - app.services.user_service - user_service - 📊 [DB QUERY] Fetching user 2 from datab
ase
app-1  | 2025-12-15 11:18:05,515 INFO sqlalchemy.engine.Engine SELECT users.id AS users_id, users.username AS users_username,
 users.email AS users_email, users.description AS users_description, users.created_at AS users_created_at, users.updated_at A
S users_updated_at
```

```
(.venv) C:\Users\highs\OneDrive\Документы\GitHub\lab6>docker start redis
redis

(.venv) C:\Users\highs\OneDrive\Документы\GitHub\lab6>curl "http://localhost:8000/users/2"
{"id":2,"username":"petrov","email":"petrov@example.com","description":"Программист и геймер","created_at":"2025-12-15T11:11:
24.350379","updated_at":"2025-12-15T11:11:25.251952"}
```

```
app-1  | LIMIT ? OFFSET ?
app-1  | INFO - 2025-12-15 11:18:05,516 - sqlalchemy.engine.Engine - base - [cached since 143.9s ago] (2, 1, 0)
app-1  | 2025-12-15 11:18:05,517 INFO sqlalchemy.engine.Engine ROLLBACK
app-1  | INFO:     172.18.0.1:60658 - "GET /users/2 HTTP/1.1" 200 OK
app-1  | INFO - 2025-12-15 11:18:05,517 - sqlalchemy.engine.Engine - base - ROLLBACK
app-1  | INFO - 2025-12-15 11:19:09,057 - app.services.user_service - user_service - 🟢 [CACHE HIT] User 2 fo und in Redis

app-1  | INFO:     172.18.0.1:42720 - "GET /users/2 HTTP/1.1" 200 OK
```

Итак, включение и отключение redis не влияет на работу всего приложения.

Теперь сделаем аналогичную обработку с продуктами.

```python
    async def get_product(self, session: Session, product_id: int) -> Optional[Product]:
        """Получить продукт по ID с кешированием на 10 минут"""
        redis_client = get_redis()

        # Пробуем получить из кеша
        if redis_client:
            cached_data = redis_client.get(self._cache_key(product_id))
            if cached_data:
                logger.info(f"🟢 [CACHE HIT] Product {product_id} found in Redis")
                try:
                    data = json.loads(cached_data)
                    return self._dict_to_product(data)
                except (json.JSONDecodeError, KeyError) as e:
                    logger.warning(
                        f"🔴 [CACHE ERROR] Invalid cache for product {product_id}: {e}"
                    )
                    # Удаляем повреждённый кеш
                    try:
                        redis_client.delete(self._cache_key(product_id))
                    except (json.JSONDecodeError, ValueError):
                        pass
            else:
                logger.info(f"🟡 [CACHE MISS] Product {product_id} not in Redis")
        else:
            logger.info(f"🔵 [NO REDIS] Redis not available for product {product_id}")

        # Получаем из БД
        logger.info(f"📊 [DB QUERY] Fetching product {product_id} from database")
        product = await self.product_repository.get_by_id(session, product_id)

        # Сохраняем в кеш
        if product and redis_client:
            logger.info(f"💾 [CACHE SAVE] Saving product {product_id} to Redis")
            try:
                data = self._product_to_dict(product)
                redis_client.setex(
                    self._cache_key(product_id), 600, json.dumps(data)
                )  # 10 минут
                logger.info(f"✅ [CACHE SAVED] Product {product_id} cached for 600s")
            except (json.JSONDecodeError, ValueError) as e:
                logger.warning(
                    f"❌ [CACHE FAILED] Failed to cache product {product_id}: {e}"
                )

        return product
```

Проверим ее так же.

```
(.venv) C:\Users\highs\OneDrive\Документы\GitHub\lab6>curl "http://localhost:8000/products/5"
{"id":5,"name":"Смартфон Galaxy S24","description":"Флагманский смартфон","price":"90000.00","stock_quantity":10,"created_at"
:"2025-12-15T11:41:30.510145","updated_at":"2025-12-15T11:41:30.510146"}
(.venv) C:\Users\highs\OneDrive\Документы\GitHub\lab6>docker-compose logs app --tail=50
```

```
app-1  | INFO - 2025-12-15 11:48:36,581 - app.services.product_service - product_service - 📊 [DB QUERY] Fetching product 5 f
rom database
app-1  | 2025-12-15 11:48:36,583 INFO sqlalchemy.engine.Engine BEGIN (implicit)
app-1  | 2025-12-15 11:48:36,586 INFO sqlalchemy.engine.Engine SELECT products.id AS products_id, products.name AS products_n
ame, products.description AS products_description, products.price AS products_price, products.stock_quantity AS products_stoc
k_quantity, products.created_at AS products_created_at, products.updated_at AS products_updated_at
app-1  | FROM products
app-1  | WHERE products.id = ?
app-1  |  LIMIT ? OFFSET ?
app-1  | 2025-12-15 11:48:36,586 INFO sqlalchemy.engine.Engine [generated in 0.00025s] (5, 1, 0)
app-1  | INFO - 2025-12-15 11:48:36,583 - sqlalchemy.engine.Engine - base - BEGIN (implicit)
app-1  | INFO - 2025-12-15 11:48:36,586 - sqlalchemy.engine.Engine - base - SELECT products.id AS products_id, products.name
AS products_name, products.description AS products_description, products.price AS products_price, products.stock_quantity AS
products_stock_quantity, products.created_at AS products_created_at, products.updated_at AS products_updated_at
app-1  | FROM products
app-1  | WHERE products.id = ?
app-1  |  LIMIT ? OFFSET ?
app-1  |  LIMIT ? OFFSET ?
app-1  |  LIMIT ? OFFSET ?
app-1  |  LIMIT ? OFFSET ?
app-1  |  LIMIT ? OFFSET ?
app-1  |  LIMIT ? OFFSET ?
app-1  |  LIMIT ? OFFSET ?
app-1  |  LIMIT ? OFFSET ?
app-1  |  LIMIT ? OFFSET ?
app-1  | INFO - 2025-12-15 11:48:36,586 - sqlalchemy.engine.Engine - base - [generated in 0.00025s] (5, 1, 0)
app-1  | INFO - 2025-12-15 11:48:36,588 - app.services.product_service - product_service - 💾 [CACHE SAVE] Saving product 5 t
o Redis
app-1  | INFO - 2025-12-15 11:48:36,588 - app.services.product_service - product_service - ✅ [CACHE SAVED] Product 5 cached
 fr 600s
```

```
app-1  | INFO - 2025-12-15 11:50:22,760 - app.services.product_service - product_service - 🟢 [CACHE HIT] Pro duct 5 found in

Redis
app-1  | INFO:     172.18.0.1:32868 - "GET /products/5 HTTP/1.1" 200 OK
```

Теперь проверим обновление продукта

```
(.venv) C:\Users\highs\OneDrive\Документы\GitHub\lab6>curl -X PUT "http://localhost:8000/products/5" -H "Content-Type: applic
ation/json" -d "{\"price\":149.99}"
{"id":5,"name":"Смартфон Galaxy S24","description":"Флагманский смартфон","price":"149.99","stock_quantity":10,"created_at":"
2025-12-15T11:41:30.510145","updated_at":"2025-12-15T11:51:17.620772"}
```

```
app-1  | INFO:     172.18.0.1:58718 - "PUT /products/5 HTTP/1.1" 200 OK
app-1  | 2025-12-15 11:51:28,389 INFO sqlalchemy.engine.Engine BEGIN (implicit)
app-1  | 2025-12-15 11:51:28,390 INFO sqlalchemy.engine.Engine SELECT products.id AS products_id, products.name AS products_n
ame, products.description AS products_description, products.price AS products_price, products.stock_quantity AS products_stoc
k_quantity, products.created_at AS products_created_at, products.updated_at AS products_updated_at
app-1  | FROM products
app-1  | WHERE products.id = ?
app-1  |  LIMIT ? OFFSET ?
app-1  | INFO - 2025-12-15 11:51:28,389 - app.services.product_service - product_service - 🟡 [CACHE MISS] Pr oduct 5 not in
R
edis
app-1  | INFO - 2025-12-15 11:51:28,389 - app.services.product_service - product_service - 📊 [DB QUERY] Fetching product 5 f
rom database
app-1  | 2025-12-15 11:51:28,390 INFO sqlalchemy.engine.Engine [cached since 171.8s ago] (5, 1, 0)
app-1  | INFO - 2025-12-15 11:51:28,389 - sqlalchemy.engine.Engine - base - BEGIN (implicit)
app-1  | INFO - 2025-12-15 11:51:28,390 - sqlalchemy.engine.Engine - base - SELECT products.id AS products_id, products.name
AS products_name, products.description AS products_description, products.price AS products_price, products.stock_quantity AS
products_stock_quantity, products.created_at AS products_created_at, products.updated_at AS products_updated_at
app-1  | FROM products
app-1  | WHERE products.id = ?
app-1  |  LIMIT ? OFFSET ?
app-1  | INFO - 2025-12-15 11:51:28,390 - sqlalchemy.engine.Engine - base - [cached since 171.8s ago] (5, 1, 0)
app-1  | INFO - 2025-12-15 11:51:28,394 - app.services.product_service - product_service - 💾 [CACHE SAVE] Saving product 5 t
o Redis
app-1  | INFO - 2025-12-15 11:51:28,395 - app.services.product_service - product_service - ✅ [CACHE SAVED] Product 5 cached
f
or 600s
app-1  | 2025-12-15 11:51:28,396 INFO sqlalchemy.engine.Engine ROLLBACK
app-1  | INFO - 2025-12-15 11:51:28,396 - sqlalchemy.engine.Engine - base - ROLLBACK
app-1  | INFO:     172.18.0.1:56680 - "GET /products/5 HTTP/1.1" 200 OK
```

Также проверим TTL кэша для продукта, он стал чуть меньше, так как прошло время после команды обновления.

```
(.venv) C:\Users\highs\OneDrive\Документы\GitHub\lab6>docker exec -it redis redis-cli KEYS "product:*"
1) "product:5"

(.venv) C:\Users\highs\OneDrive\Документы\GitHub\lab6>docker exec -it redis redis-cli TTL "product:5"
(integer) 510
```

Ответы на вопросы

1. В чем заключается основное преимущество хранения данных в оперативной памяти (in-memory) по сравнению с дисковыми БД?

Оперативная память в разы быстрее дисков это обеспечивает микросекундную задержку для чтения/записи.

2. Для чего нужен параметр decode_responses=True при создании клиента Redis?

decode_responses=True автоматически декодирует байтовые ответы Redis в строки Python. Без этого все значения возвращаются как b'string'.

3. Что такое TTL (Time To Live) ключа и как он используется в Redis?

TTL – время жизни ключа в секундах. После истечения Redis автоматически удаляет ключ.

4. Объясните разницу между командами r.lpush() и r.rpush() для списков.

lpush() добавляет элемент в начало списка (слева), rpush() – в конец (справа). Определяют порядок обхода элементов списка.

5. Как обеспечить атомарность операций в Redis?

Транзакции через MULTI/EXEC или pipeline(). Lua-скрипты выполняются атомарно. Команды SETNX, INCR также атомарны.

6. Как в Redis реализована репликация и кластеризация?

Репликация: master-slave асинхронное копирование данных. Кластеризация: шардирование через слоты (16384 слота), автоматическая отказоустойчивость и миграция данных между нодами

ЗАКЛЮЧЕНИЕ

В ходе работы были освоены базовые навыки установки, подключения и взаимодействия с Redis в Python. Изучены основные структуры данных Redis и их применение на практике.