Celery + Redis + MongoDB Application

Index

1. Project Overview

This application demonstrates a production-style asynchronous backend using Flask for HTTP APIs, MongoDB for persistence, Redis as a message broker, and Celery workers for background processing.

2. Learning Outcomes

- Understand async processing using queues
- Understand separation of API and worker responsibilities
- Understand Redis as a broker, not a database
- Safely update persistent state using MongoDB

3. Scope

In scope: API → queue → worker → DB flow.
Out of scope: retries, dead-letter queues, monitoring.

4. Tech Stack & Versions

• Python 3.x
• Flask
• MongoDB
• Redis
• Celery
• Docker

5. Folder Structure

```
celery_redis_mongo_app/
├── app/
│   ├── app.py
│   ├── db.py
│   ├── celery_app.py
│   └── tasks.py
├── docker-compose.yml
├── requirements.txt
└── README.md
```

6. High-Level Architecture

```
Client → Flask API → Celery → Redis → Worker → MongoDB
```

7. Request Lifecycle

```
1. Client sends request
2. API validates and stores data
3. API enqueues task
4. Worker processes task
5. Worker updates MongoDB
```

8. API Contract

```
POST /items
GET  /items/{id}
```

9. Configuration & Environment Variables

```
MONGO_URI
REDIS_URL
CELERY_BROKER_URL
```

10. How to Run

```
docker-compose up -d
python app/app.py
python -m celery -A app.celery_app worker --pool=solo --loglevel=info
```

11. Common Mistakes

- Running API without worker
- Wrong broker URL
- Using Redis as database
- Blocking long operations inside routes

12. Debugging Techniques

- Check Redis and Mongo containers
- Verify Celery task registration
- Run worker with --loglevel=info

Appendix A: Source Code

celery_redis_mongo_app/README.md

```
# Celery + Redis + MongoDB Join Application

## Description
Asynchronous pipeline where:
- Client submits request
- Celery worker performs MongoDB join
- Result stored in Redis KV store

## Files
- celery_app.py : Celery configuration
- tasks.py : Worker task
- client.py : Submit task
- fetch_result.py : Fetch result from Redis
```

celery_redis_mongo_app/celery_app.py

```
from celery import Celery

celery_app = Celery(
    "join_tasks",
    broker="redis://localhost:6379/0",
    backend="redis://localhost:6379/1",
)

celery_app.conf.update(
    task_serializer="json",
    result_serializer="json",
    accept_content=["json"],
    task_acks_late=True,
    worker_prefetch_multiplier=1,
)

import tasks
```

celery_redis_mongo_app/client.py

```
from tasks import join_users_orders

task = join_users_orders.delay("u1")
print("Submitted task ID:", task.id)
```

celery_redis_mongo_app/fetch_result.py

```
import redis, json

r = redis.Redis(
    host="localhost",
    port=6379,
    db=2,
    decode_responses=True
```

```
)

data = r.get("user_order_result:u1")
print(json.loads(data))
```

celery_redis_mongo_app/requirements.txt

```
celery
redis
pymongo
```

celery_redis_mongo_app/tasks.py

```
from celery_app import celery_app
from pymongo import MongoClient
import redis, json

mongo = MongoClient("mongodb://localhost:27017")
db = mongo.shop

redis_client = redis.Redis(
    host="localhost",
    port=6379,
    db=2,
    decode_responses=True
)

@celery_app.task(bind=True)
def join_users_orders(self, user_id):
    pipeline = [
        {"$match": {"_id": user_id}},
        {
            "$lookup": {
                "from": "orders",
                "localField": "_id",
                "foreignField": "userId",
                "as": "orders",
            }
        },
    ]
    result = list(db.users.aggregate(pipeline))
    redis_key = f"user_order_result:{user_id}"
    redis_client.set(redis_key, json.dumps(result), ex=300)
    return {"redis_key": redis_key, "count": len(result)}
```