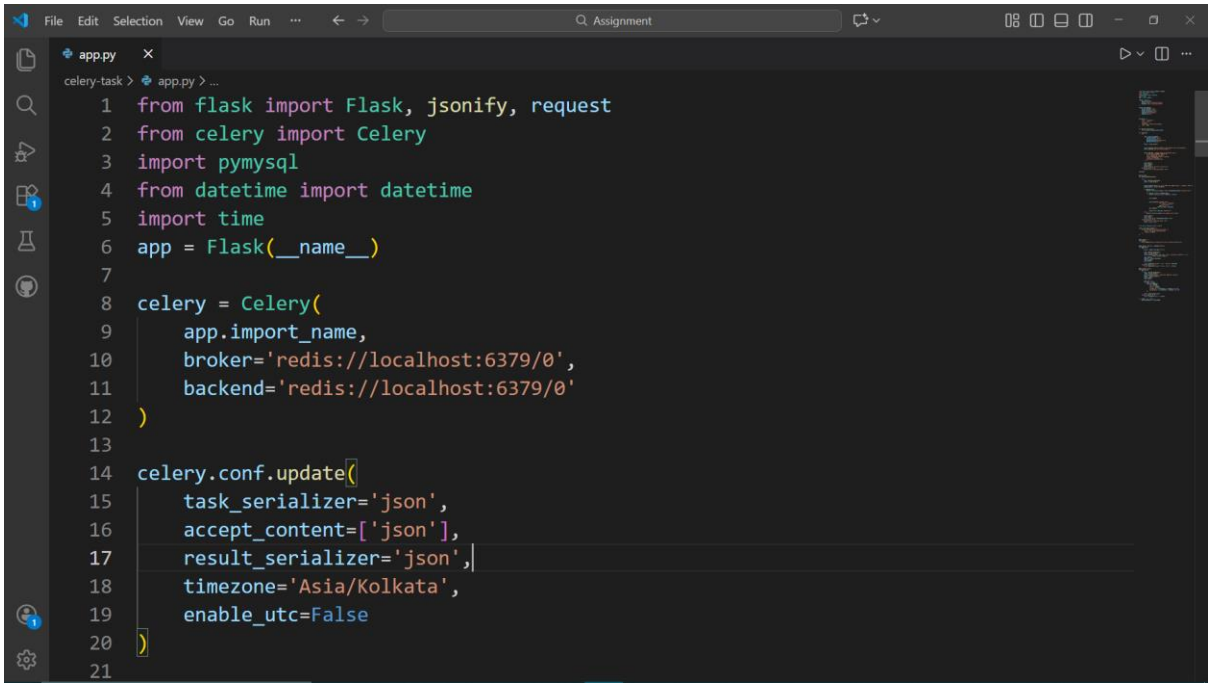
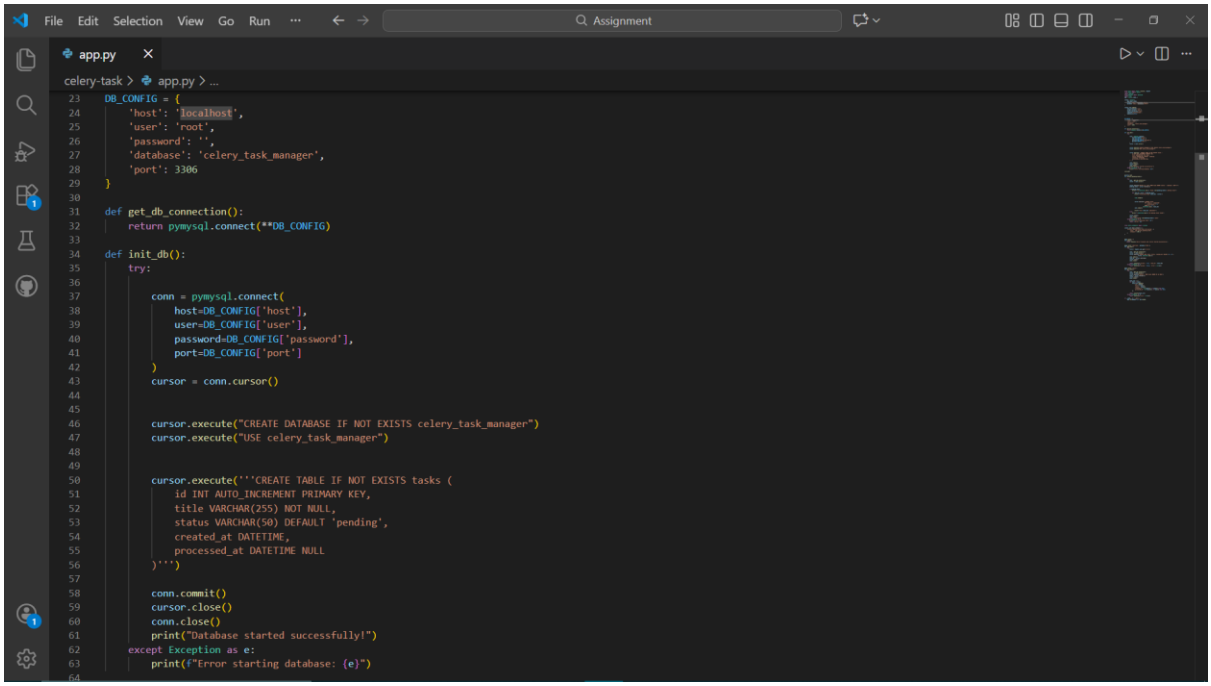
*This application demonstrates asynchronous background processing using **Flask, Celery, Redis, and MySQL**. Flask handles API requests, while Celery runs background tasks independently. Redis is used as the message broker, and MySQL stores task data.*

1.Celery Setup



Celery is configured with Redis as both the broker and result backend. JSON serialization is used for lightweight communication. The timezone is set to Asia/Kolkata to ensure consistent scheduling and logging.

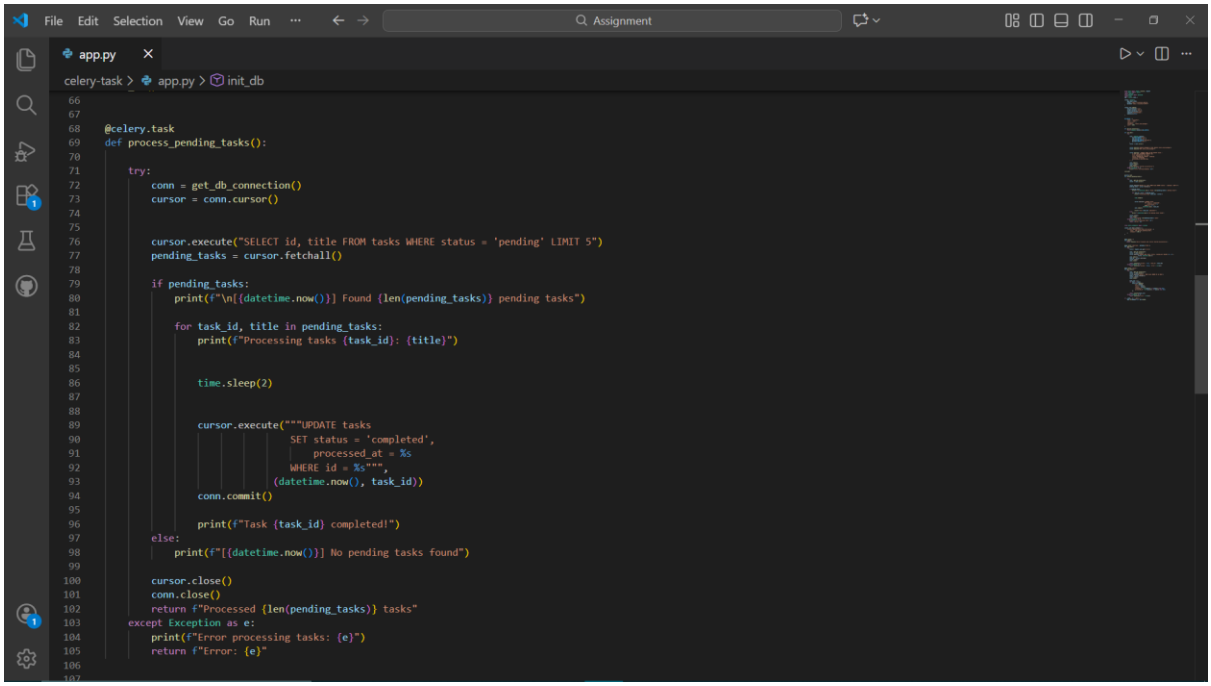
2.Database Initialization



```python
DB_CONFIG = {
    'host': 'localhost',
    'user': 'root',
    'password': '',
    'database': 'celery_task_manager',
    'port': 3306
}

def get_db_connection():
    return pymysql.connect(**DB_CONFIG)

def init_db():
    try:

        conn = pymysql.connect(
            host=DB_CONFIG['host'],
            user=DB_CONFIG['user'],
            password=DB_CONFIG['password'],
            port=DB_CONFIG['port']
        )
        cursor = conn.cursor()


        cursor.execute("CREATE DATABASE IF NOT EXISTS celery_task_manager")
        cursor.execute("USE celery_task_manager")


        cursor.execute('''CREATE TABLE IF NOT EXISTS tasks (
            id INT AUTO_INCREMENT PRIMARY KEY,
            title VARCHAR(255) NOT NULL,
            status VARCHAR(50) DEFAULT 'pending',
            created_at DATETIME,
            processed_at DATETIME NULL
        )''')

        conn.commit()
        cursor.close()
        conn.close()
        print("Database started successfully!")
    except Exception as e:
        print(f"Error starting database: {e}")
```

On startup, the application creates the database and a tasks table if they do not exist. The table stores task titles, status, creation time, and processing time, allowing the system to run without manual database setup.

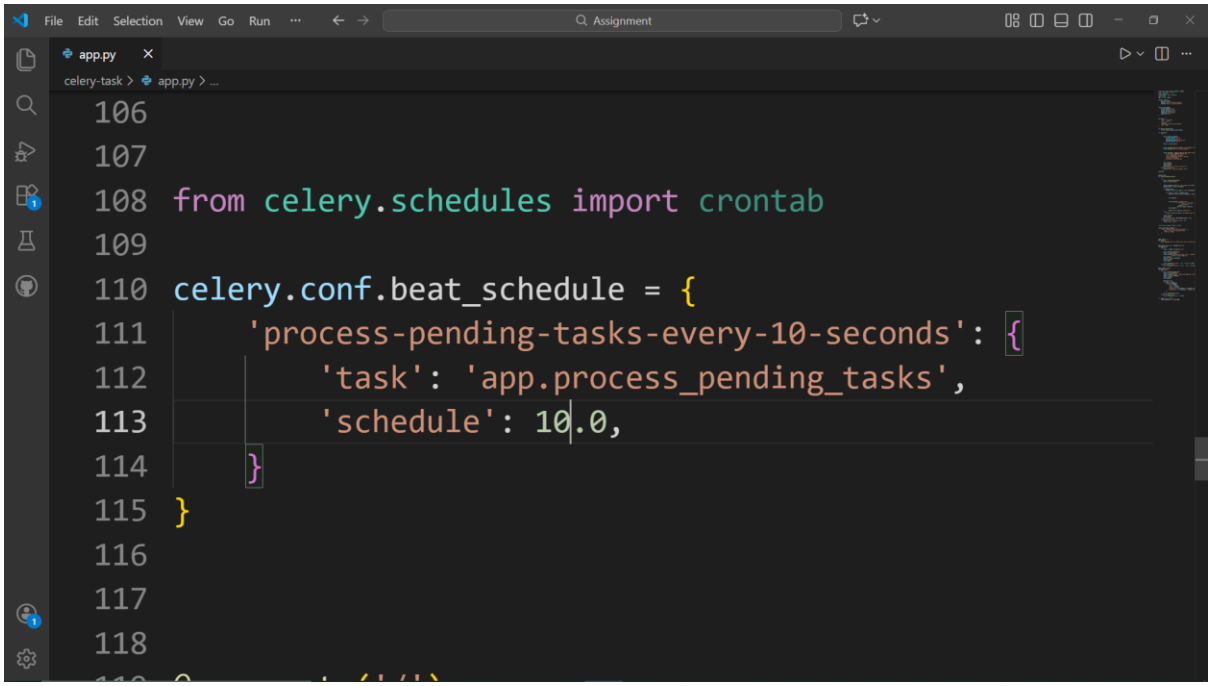
3.Background Task Logic



```python
@celery.task
def process_pending_tasks():

    try:
        conn = get_db_connection()
        cursor = conn.cursor()


        cursor.execute("SELECT id, title FROM tasks WHERE status = 'pending' LIMIT 5")
        pending_tasks = cursor.fetchall()

        if pending_tasks:
            print(f"\n[{datetime.now()}] Found {len(pending_tasks)} pending tasks")

            for task_id, title in pending_tasks:
                print(f"Processing tasks {task_id}: {title}")


                time.sleep(2)


                cursor.execute("""UPDATE tasks
                                  SET status = 'completed',
                                      processed_at = %s
                                  WHERE id = %s""",
                               (datetime.now(), task_id))
                conn.commit()

                print(f"Task {task_id} completed!")
        else:
            print(f"[{datetime.now()}] No pending tasks found")

        cursor.close()
        conn.close()
        return f"Processed {len(pending_tasks)} tasks"
    except Exception as e:
        print(f"Error processing tasks: {e}")
        return f"Error: {e}"
```

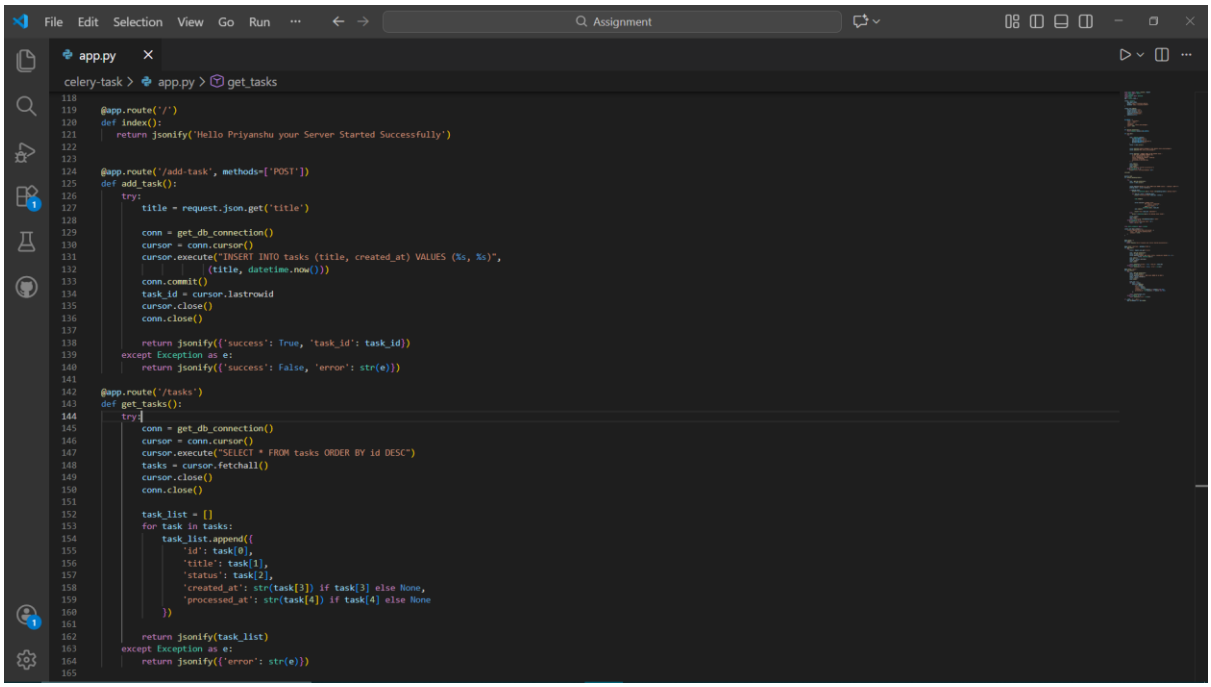
The Celery task periodically queries the database for pending tasks. When tasks are found, they are processed one by one and marked as completed with a timestamp. If no tasks are available, the worker logs the status and waits for the next cycle.

4.Periodic Scheduling



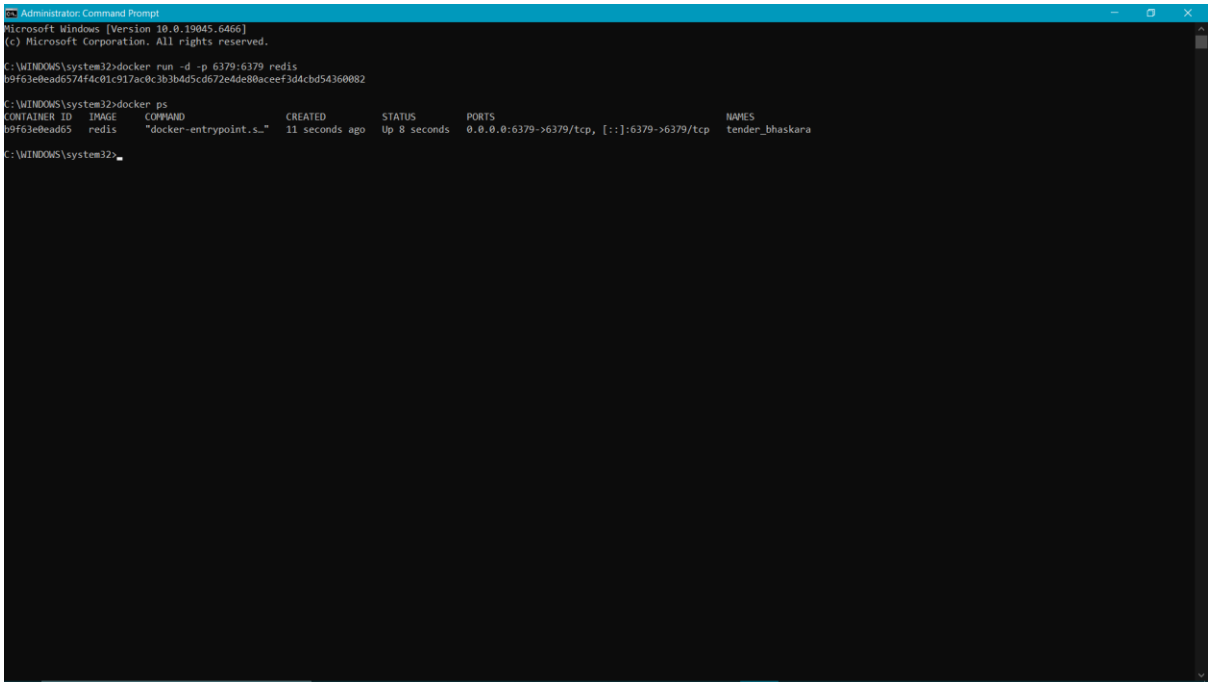
Celery Beat schedules the background task to run every fixed interval 10 seconds. This scheduling is time-based and independent of when tasks are created, ensuring predictable background execution.

5.Flask API Endpoints



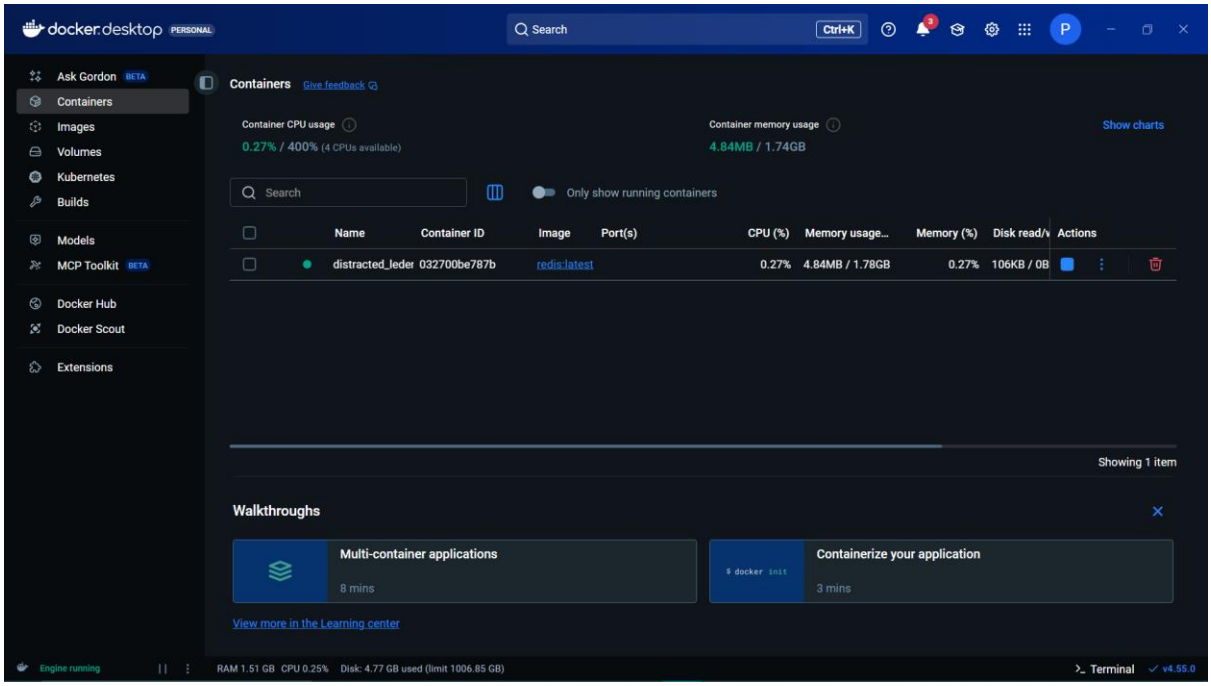
The Flask server provides endpoints to add new tasks and retrieve all tasks. These endpoints interact with the database directly while background processing happens asynchronously.

6.Starting the Application



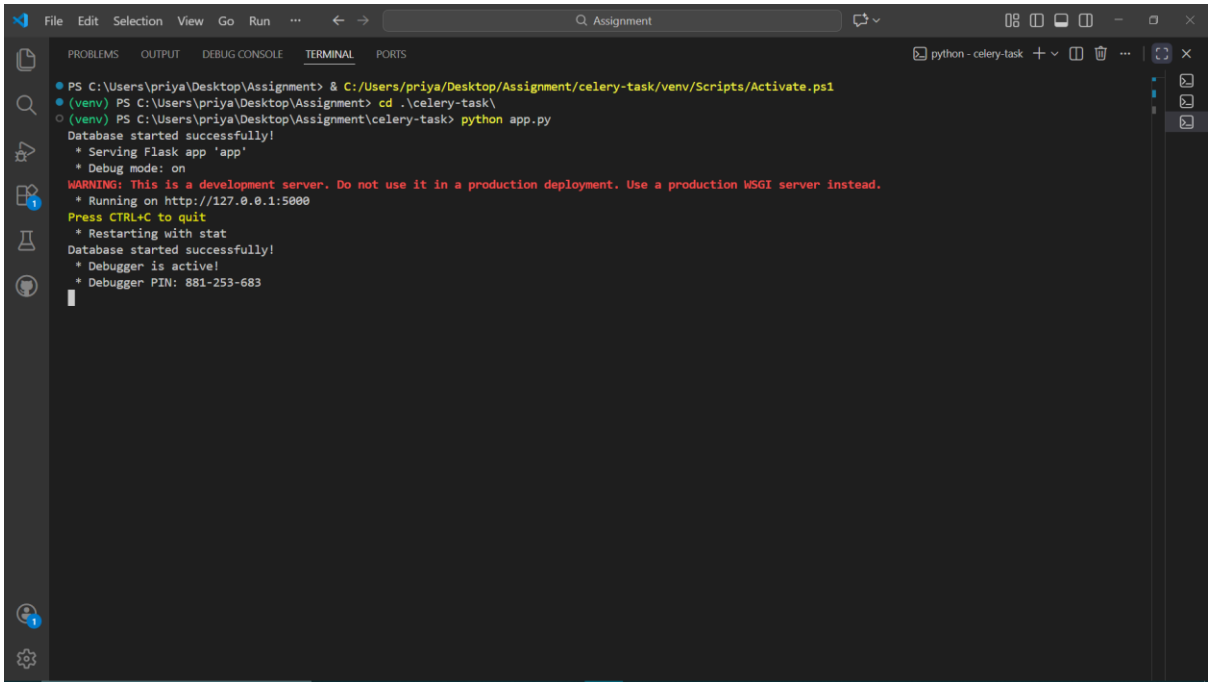
Starting the redis server in CMD using docker on port 6379

7.Checking With Docker Engine



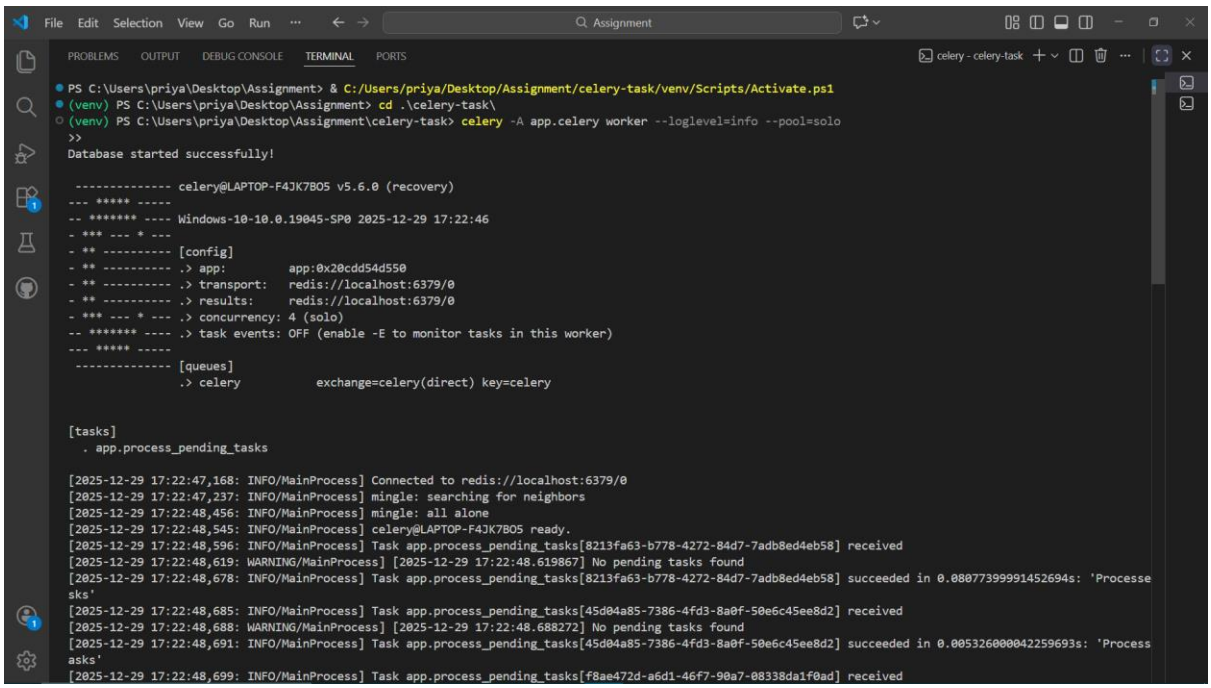
Checking for the running images to make sure redis is running

8.Starting the app.py



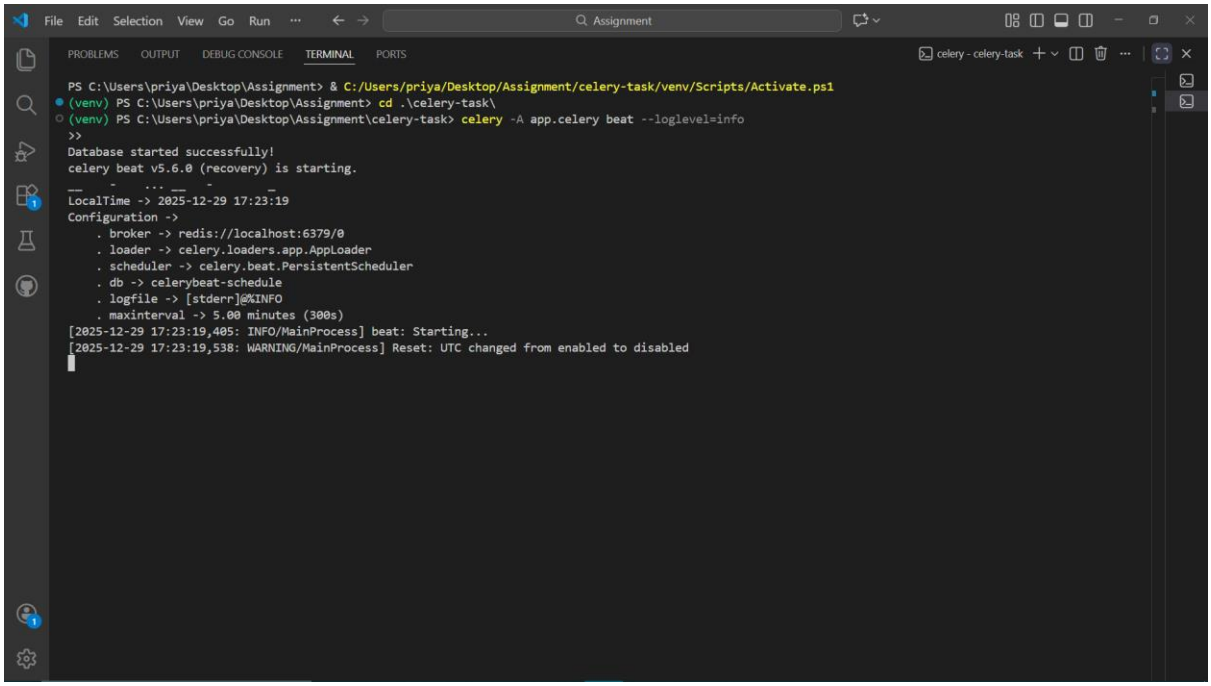
The app starts running on the port 5000 as we start the application with **python app.py**

9.Starting celery worker



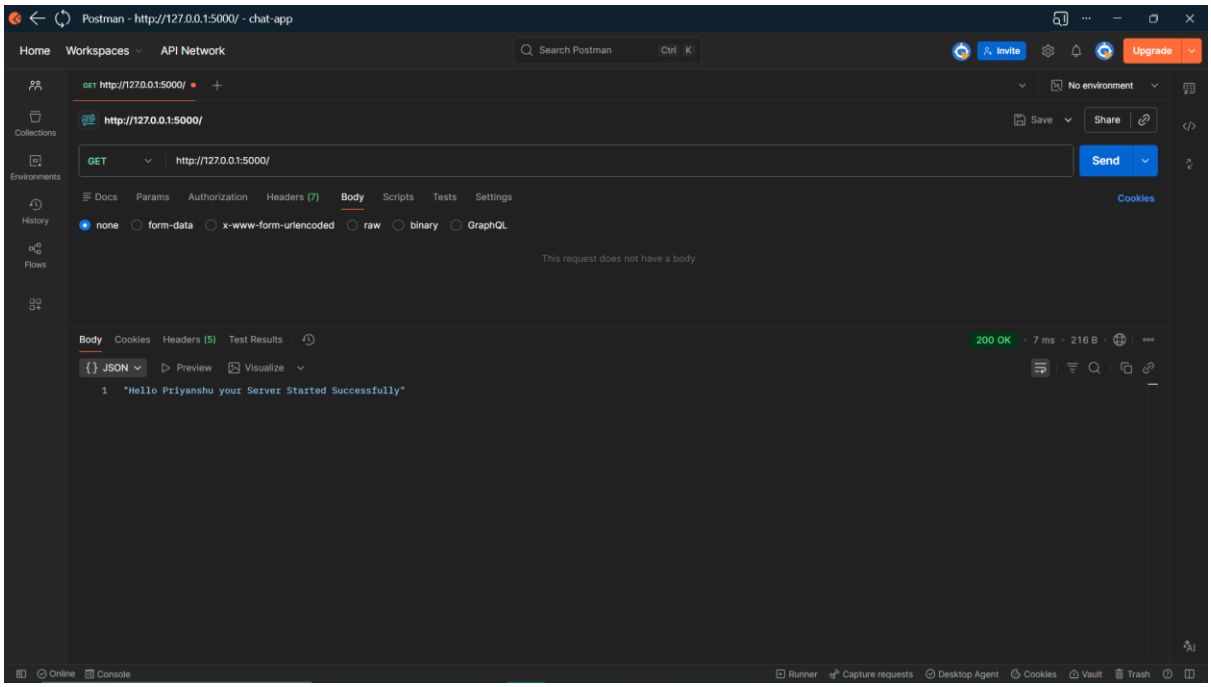
We start the celery worker in app.py using celery -A app.celery worker --loglevel=info --pool=solo

10. Celery Beat



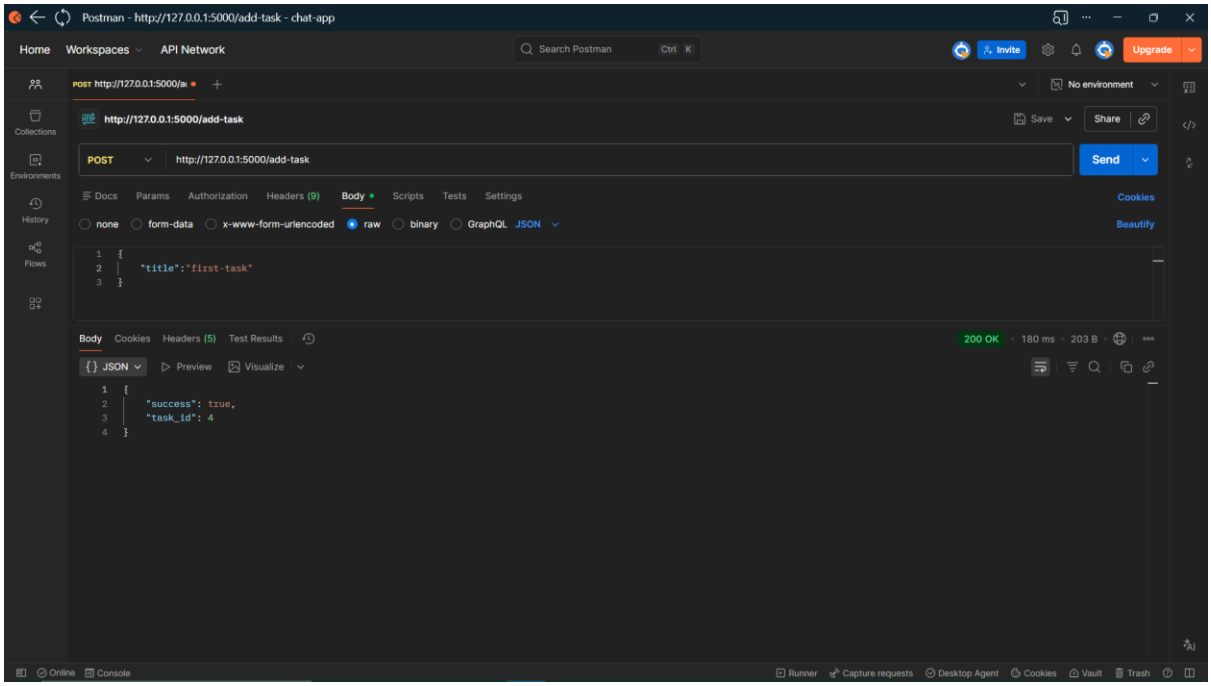
We start the celery beat using celery -A app.celery beat --loglevel=info

11.Starting the app on postman



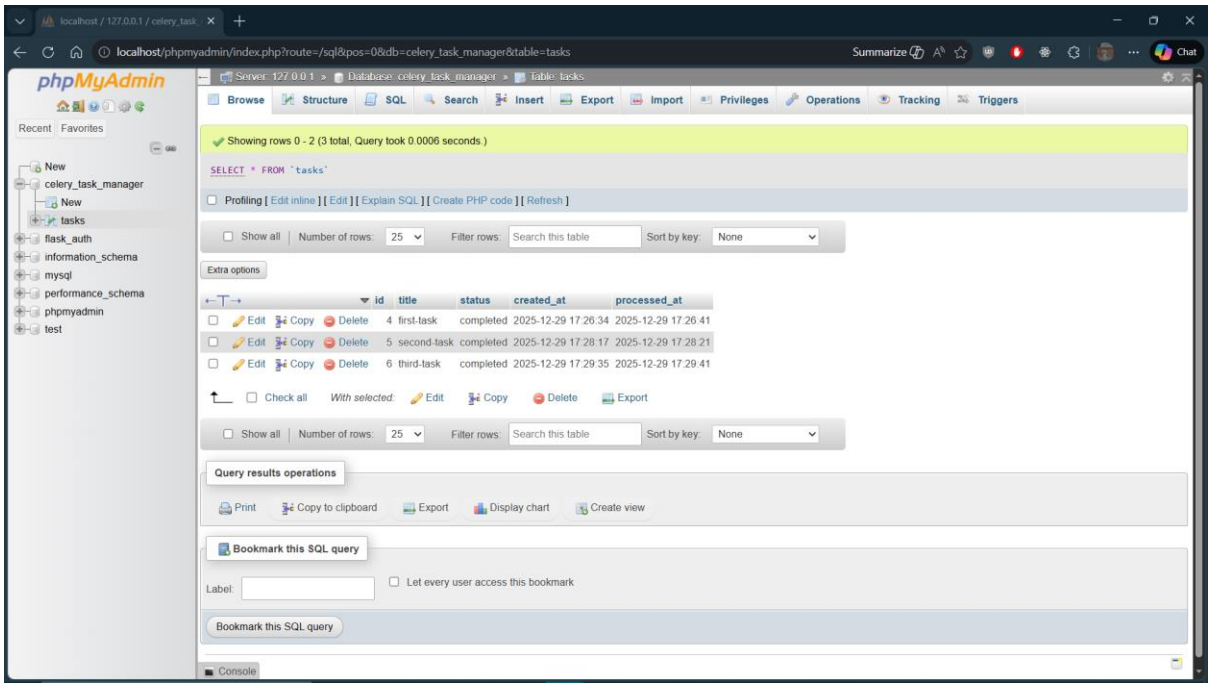
We start the application and server is Successfully Started at the (‘/’) route

12.Adding task to the DB



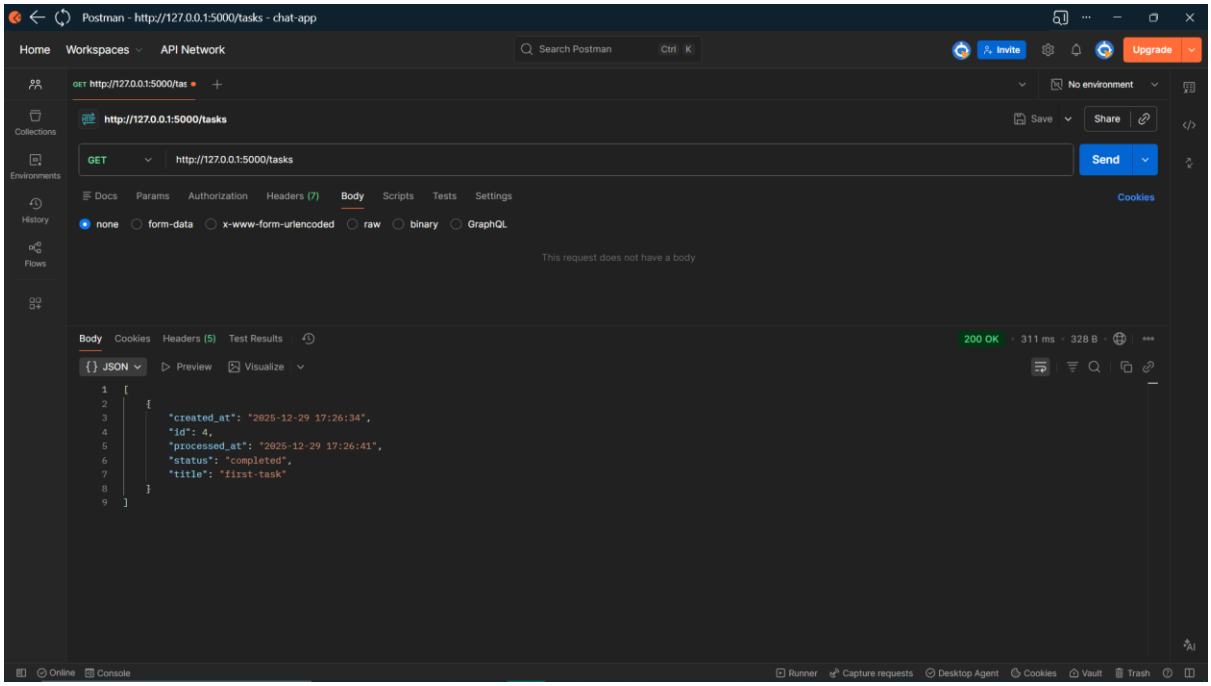
We add tasks to the db for celery to start fetching tasks with title as input at the ('/add-task') route

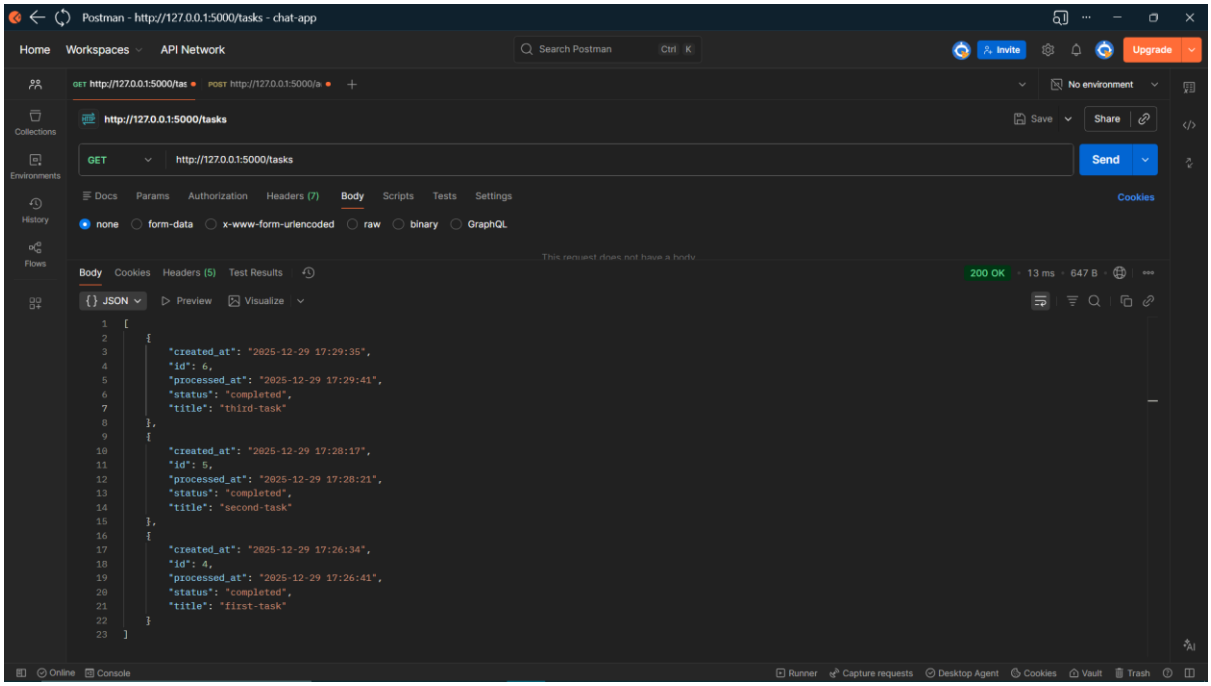
13.DB check at php-myadmin (XAMPP software)



We check if our entry is added to database by checking at the my-php-admin server

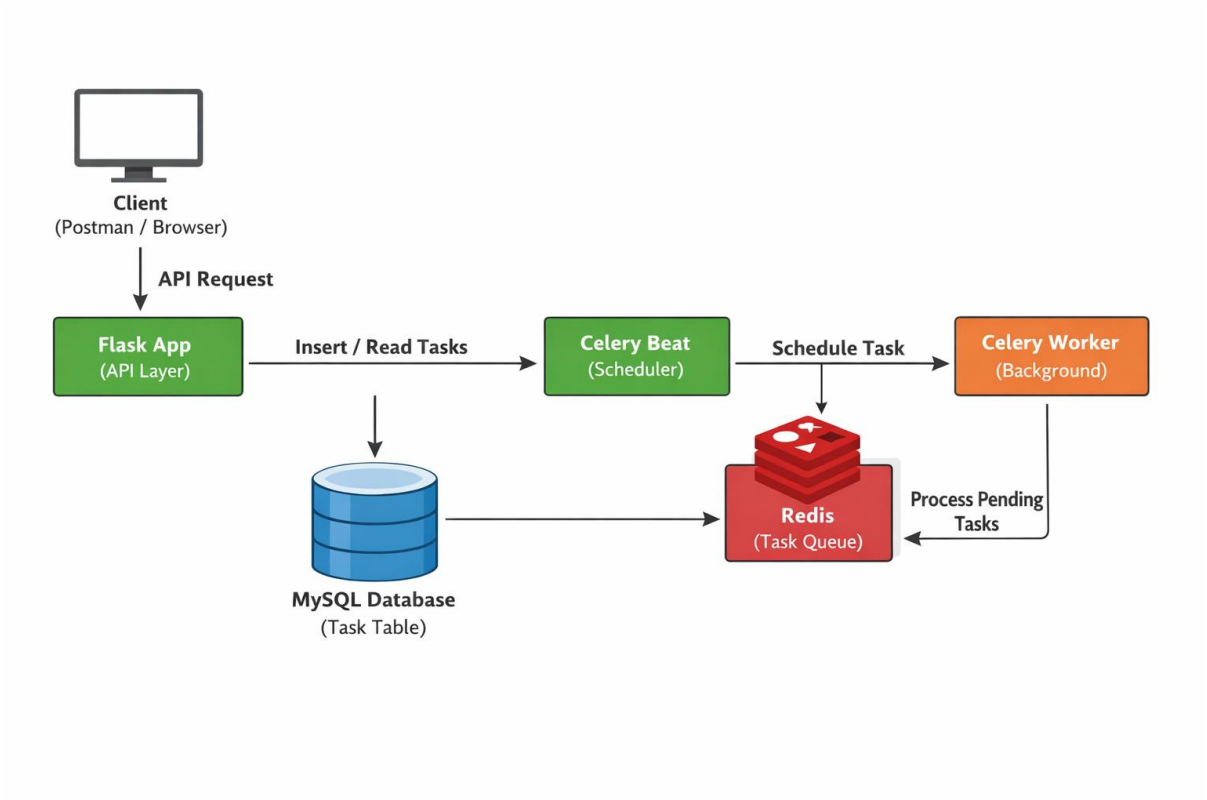
14.Check for tasks





We check for tasks at the (‘/tasks’) route to check all pending/completed tasks and we can see the status in response

15.Execution Flow:



Flask handles user requests, Redis manages task messaging, and Celery workers process tasks in the background. This separation ensures non-blocking execution and scalability