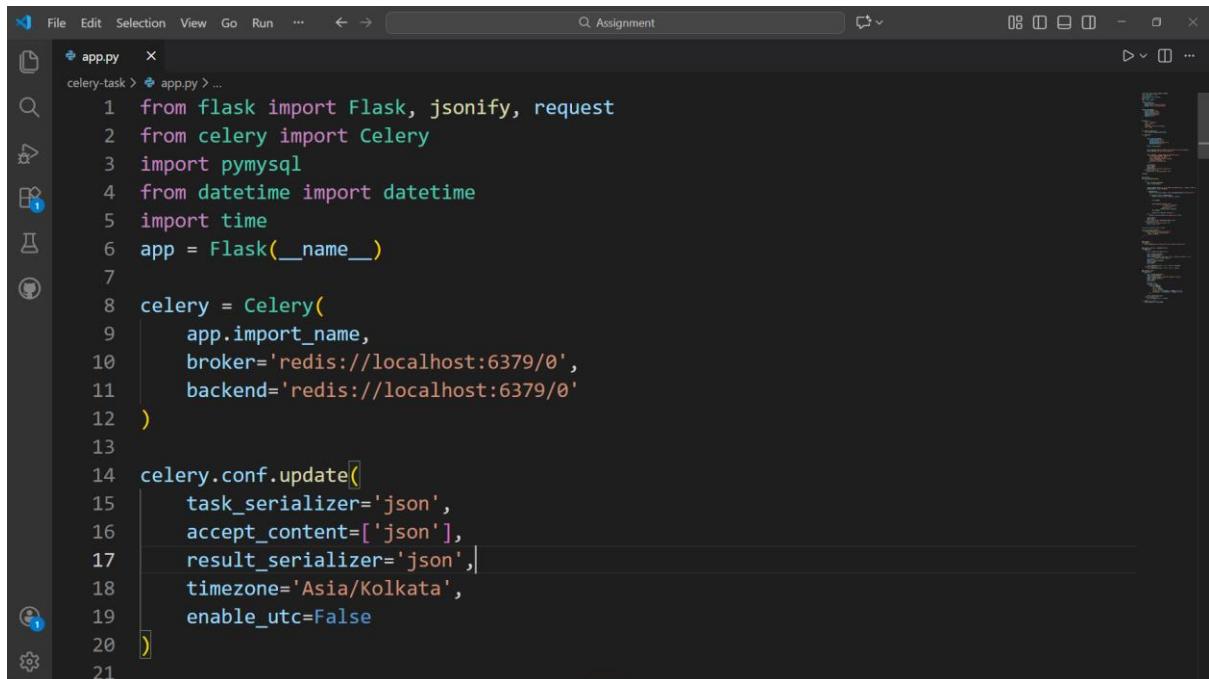


*This application demonstrates asynchronous background processing using **Flask**, **Celery**, **Redis**, and **MySQL**. Flask handles API requests, while Celery runs background tasks independently. Redis is used as the message broker, and MySQL stores task data.*

1.Celery Setup

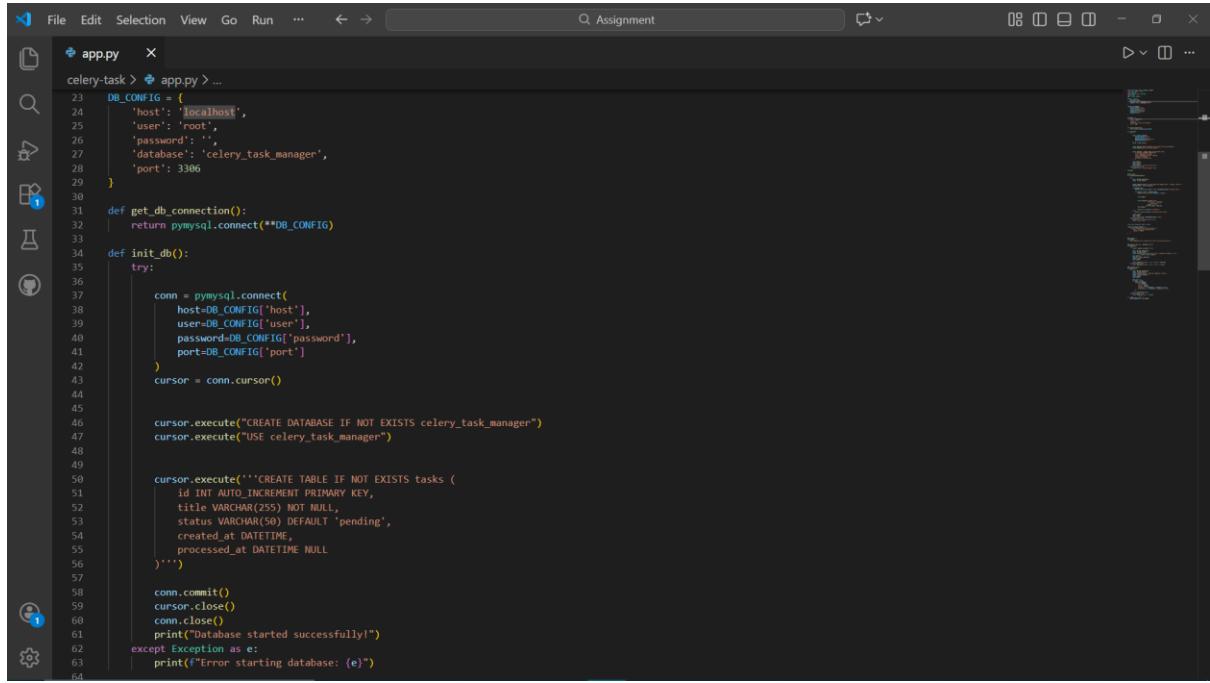


The screenshot shows a code editor window with the file 'app.py' open. The code defines a Flask application and sets up Celery with Redis as both the broker and result backend. JSON serialization is used for tasks, and the timezone is set to Asia/Kolkata.

```
File Edit Selection View Go Run ... ← → Q Assignment
app.py x
celery-task > app.py > ...
1 from flask import Flask, jsonify, request
2 from celery import Celery
3 import pymysql
4 from datetime import datetime
5 import time
6 app = Flask(__name__)
7
8 celery = Celery(
9     app.import_name,
10    broker='redis://localhost:6379/0',
11    backend='redis://localhost:6379/0'
12 )
13
14 celery.conf.update(
15     task_serializer='json',
16     accept_content=['json'],
17     result_serializer='json',
18     timezone='Asia/Kolkata',
19     enable_utc=False
20 )
21
```

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.

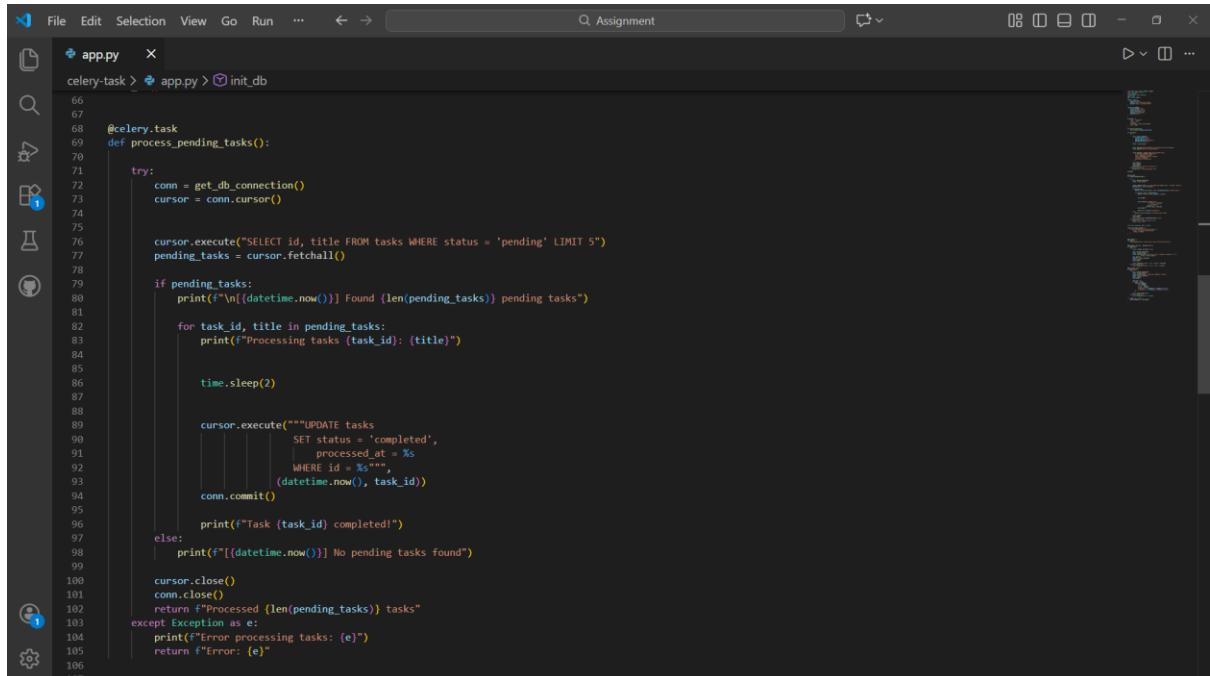
2. Database Initialization



```
celery-task > app.py > ...  
app.py  
23     DB_CONFIG = {  
24         'host': 'localhost',  
25         'user': 'root',  
26         'password': '',  
27         'database': 'celery_task_manager',  
28         'port': 3306  
29     }  
30  
31     def get_db_connection():  
32         return pymysql.connect(**DB_CONFIG)  
33  
34     def init_db():  
35         try:  
36             conn = pymysql.connect(  
37                 host=DB_CONFIG['host'],  
38                 user=DB_CONFIG['user'],  
39                 password=DB_CONFIG['password'],  
40                 port=DB_CONFIG['port'])  
41             cursor = conn.cursor()  
42  
43             cursor.execute("CREATE DATABASE IF NOT EXISTS celery_task_manager")  
44             cursor.execute("USE celery_task_manager")  
45  
46             cursor.execute('''CREATE TABLE IF NOT EXISTS tasks (  
47                 id INT AUTO_INCREMENT PRIMARY KEY,  
48                 title VARCHAR(255) NOT NULL,  
49                 status VARCHAR(50) DEFAULT 'pending',  
50                 created_at DATETIME,  
51                 processed_at DATETIME NULL  
52             )'''  
53             )  
54             conn.commit()  
55             cursor.close()  
56             conn.close()  
57             print("Database started successfully!")  
58         except Exception as e:  
59             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.

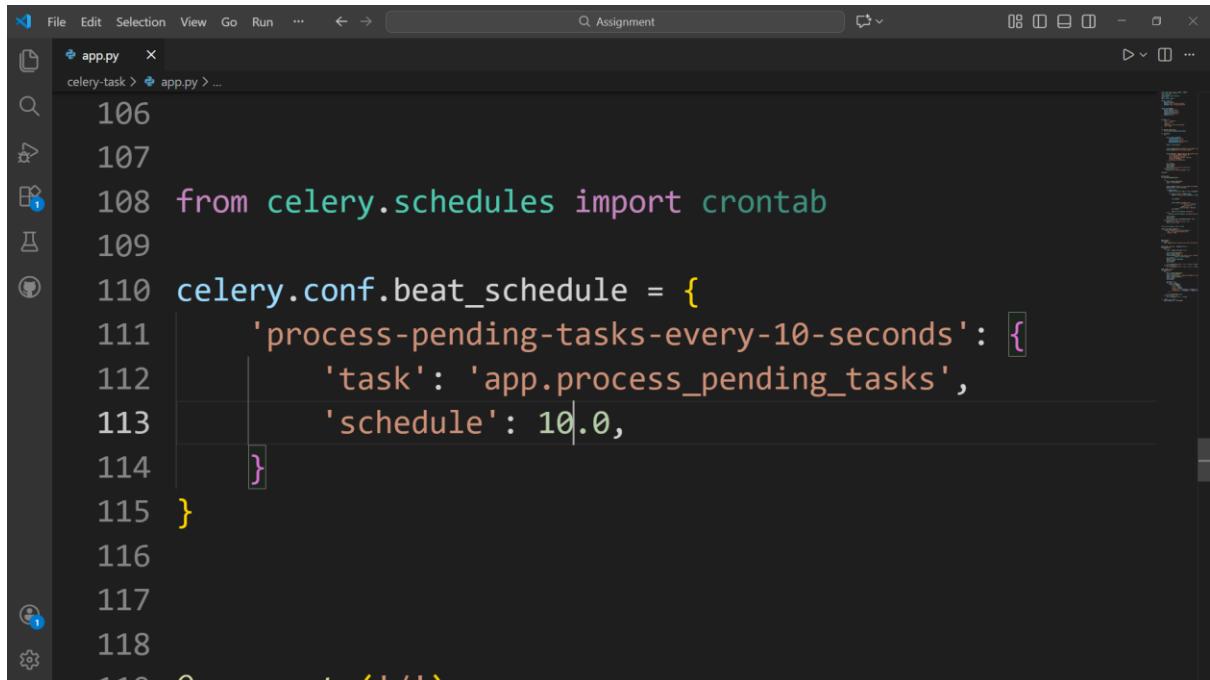
3. Background Task Logic



```
celery-task > app.py > init_db  
app.py  
66  
67  
68     @celery.task  
69     def process_pending_tasks():  
70         try:  
71             conn = get_db_connection()  
72             cursor = conn.cursor()  
73  
74             cursor.execute("SELECT id, title FROM tasks WHERE status = 'pending' LIMIT 5")  
75             pending_tasks = cursor.fetchall()  
76  
77             if pending_tasks:  
78                 print(f"\n[datetime.now()] Found {len(pending_tasks)} pending tasks")  
79  
80                 for task_id, title in pending_tasks:  
81                     print(f"Processing tasks {task_id}: {title}")  
82  
83                     time.sleep(2)  
84  
85  
86                     cursor.execute("""UPDATE tasks  
87                         SET status = 'completed',  
88                             processed_at = %s  
89                         WHERE id = %s""",  
90                         (datetime.now(), task_id))  
91             conn.commit()  
92  
93             print(f"Task {task_id} completed!")  
94         else:  
95             print(f"[datetime.now()] No pending tasks found")  
96  
97         cursor.close()  
98         conn.close()  
99         return f"Processed {len(pending_tasks)} tasks"  
100     except Exception as e:  
101         print(f"Error processing tasks: {e}")  
102         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.

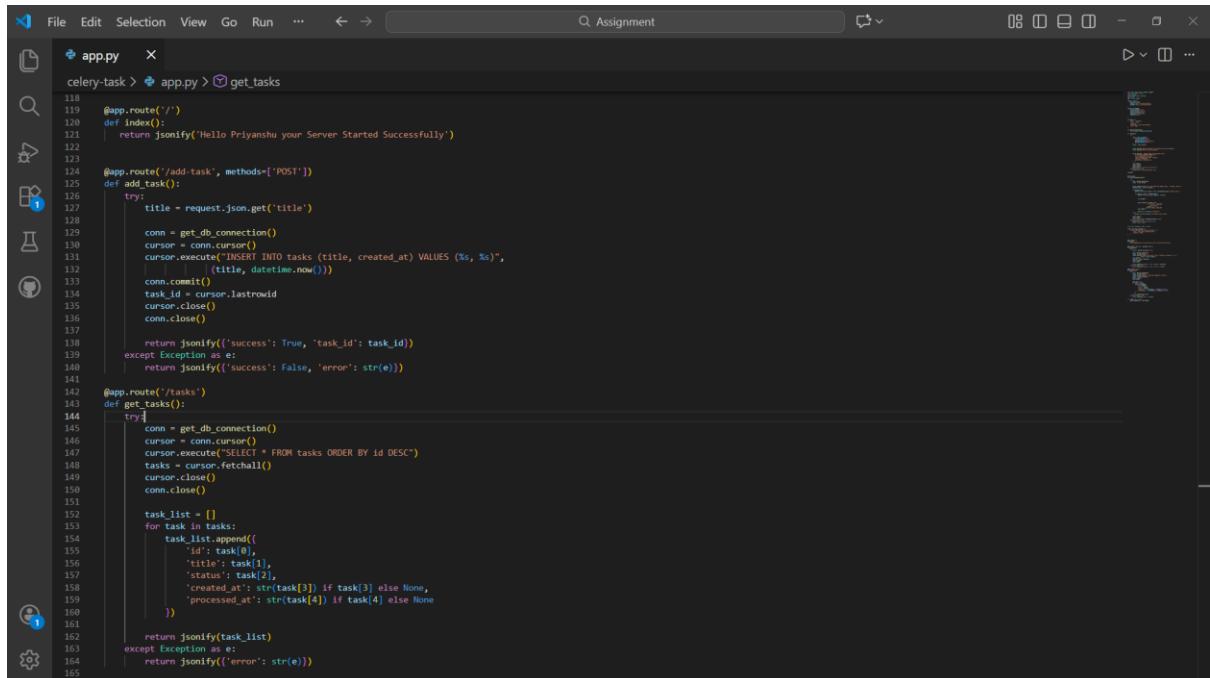
4. Periodic Scheduling



```
106
107
108 from celery.schedules import crontab
109
110 celery.conf.beat_schedule = {
111     'process-pending-tasks-every-10-seconds': {
112         'task': 'app.process_pending_tasks',
113         'schedule': 10.0,
114     }
115 }
```

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.

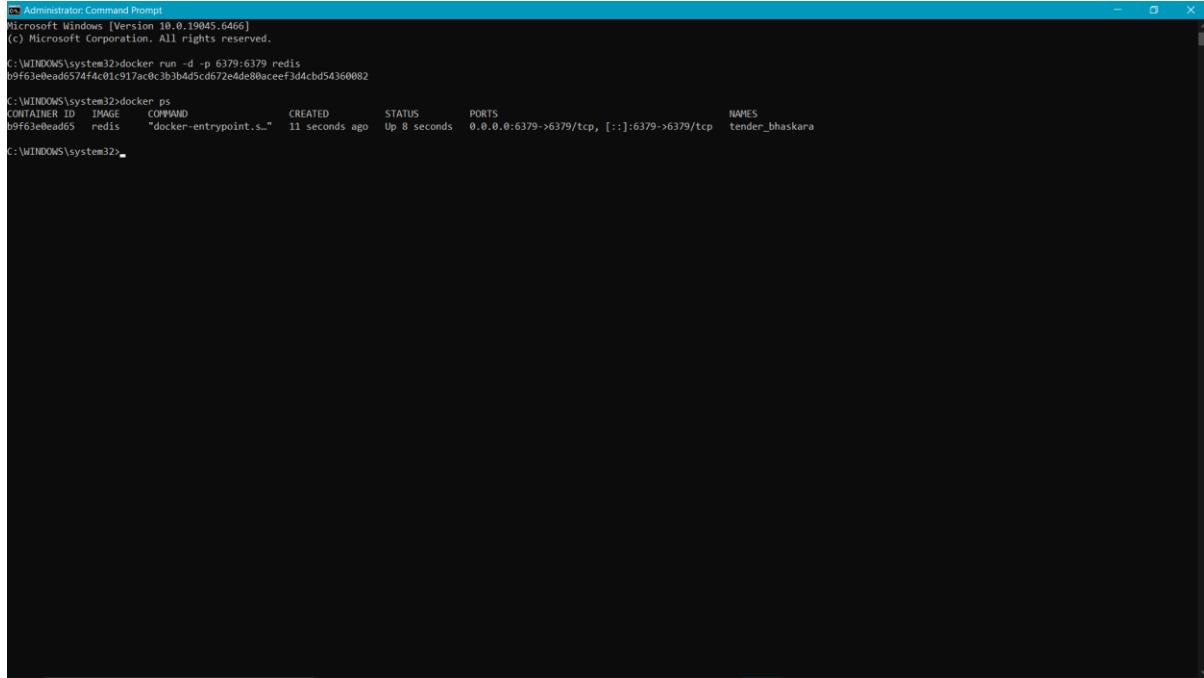
5. Flask API Endpoints



```
118
119     @app.route('/')
120     def index():
121         return jsonify({'Hello Priyanshu your Server Started Successfully'})
122
123
124     @app.route('/add-task', methods=['POST'])
125     def add_task():
126         try:
127             title = request.json.get('title')
128
129             conn = get_db_connection()
130             cursor = conn.cursor()
131             cursor.execute("INSERT INTO tasks (title, created_at) VALUES (%s, %s)",
132                           (title, datetime.now()))
133             conn.commit()
134             task_id = cursor.lastrowid
135             cursor.close()
136             conn.close()
137
138             return jsonify({'success': True, 'task_id': task_id})
139         except Exception as e:
140             return jsonify({'success': False, 'error': str(e)})
141
142     @app.route('/tasks')
143     def get_tasks():
144         try:
145             conn = get_db_connection()
146             cursor = conn.cursor()
147             cursor.execute("SELECT * FROM tasks ORDER BY id DESC")
148             tasks = cursor.fetchall()
149             cursor.close()
150             conn.close()
151
152             task_list = []
153             for task in tasks:
154                 task_list.append({
155                     'id': task[0],
156                     'title': task[1],
157                     'status': task[2],
158                     'created_at': str(task[3]) if task[3] else None,
159                     'processed_at': str(task[4]) if task[4] else None
160                 })
161
162             return jsonify(task_list)
163         except Exception as e:
164             return jsonify({'error': str(e)})
```

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



```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.19045.6466]
(c) Microsoft Corporation. All rights reserved.

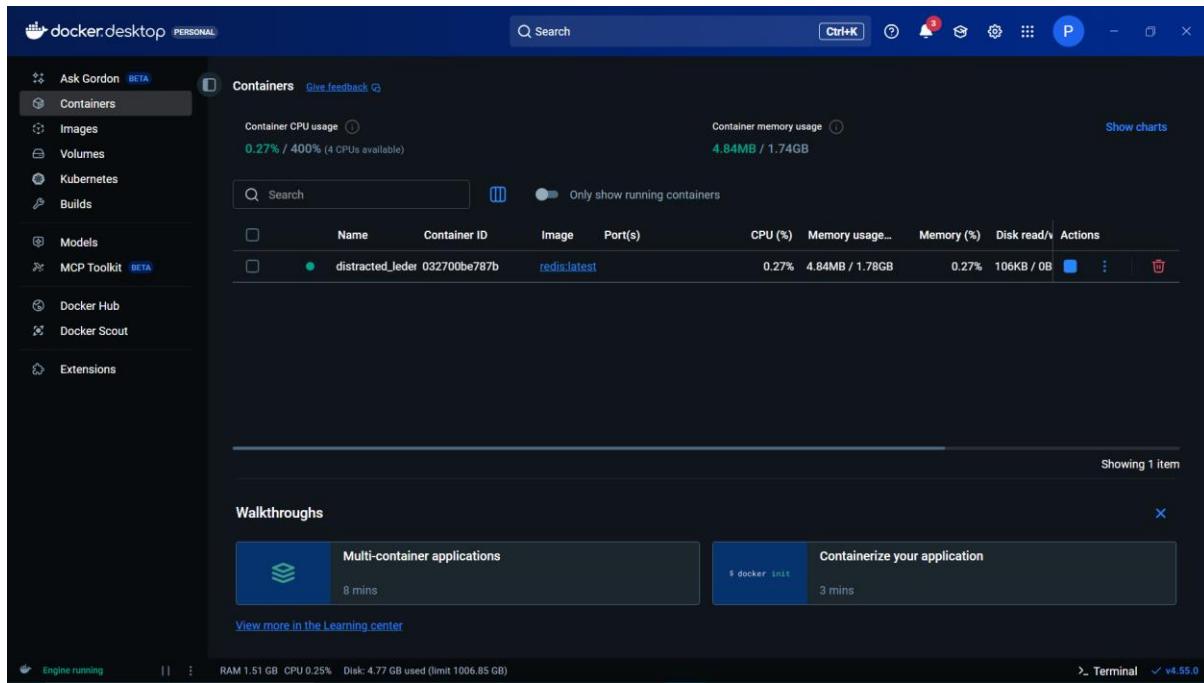
C:\WINDOWS\system32>docker run -d -p 6379:6379 redis
b9f63e0ead6574fc81c917acd3b3b4d5cd672e4de80acef3d4chd54360082

C:\WINDOWS\system32>docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
b9f63e0ead65 redis "docker-entrypoint.s..." 11 seconds ago Up 8 seconds 0.0.0.0:6379->6379/tcp, [::]:6379->6379/tcp tender_bhaskara

C:\WINDOWS\system32>
```

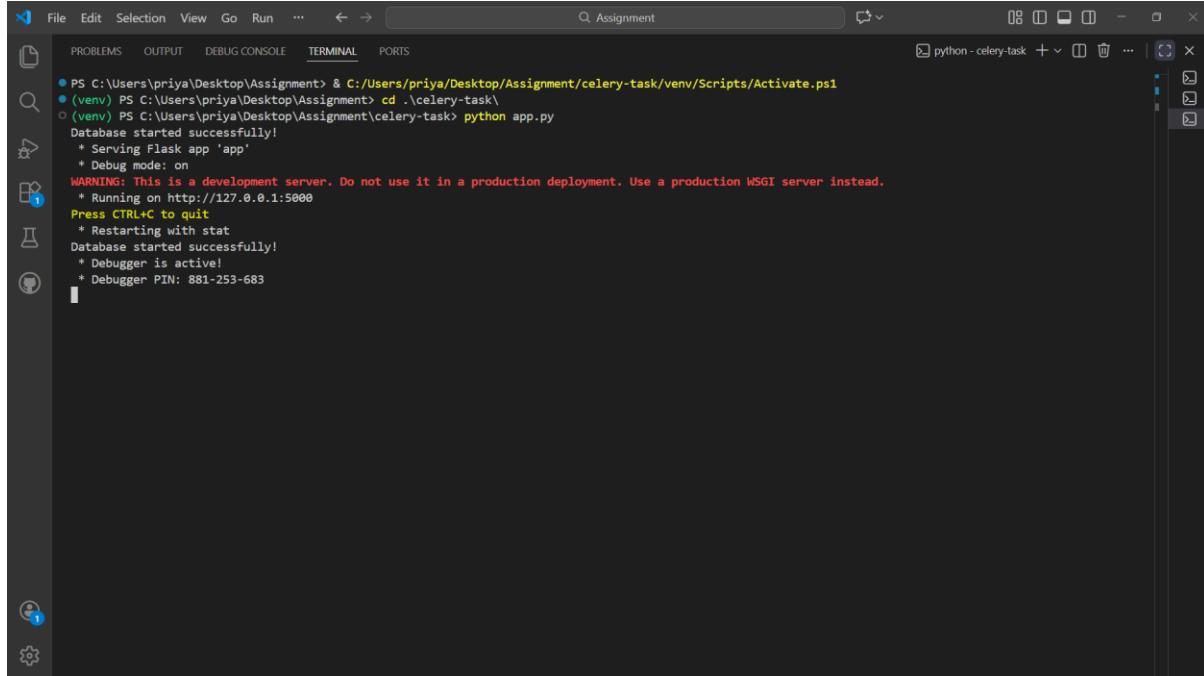
Starting the redis server in CMD using docker on port 6379

7.Checking With Docker Engine



Checking for the running images to make sure redis is running

8.Starting the app.py

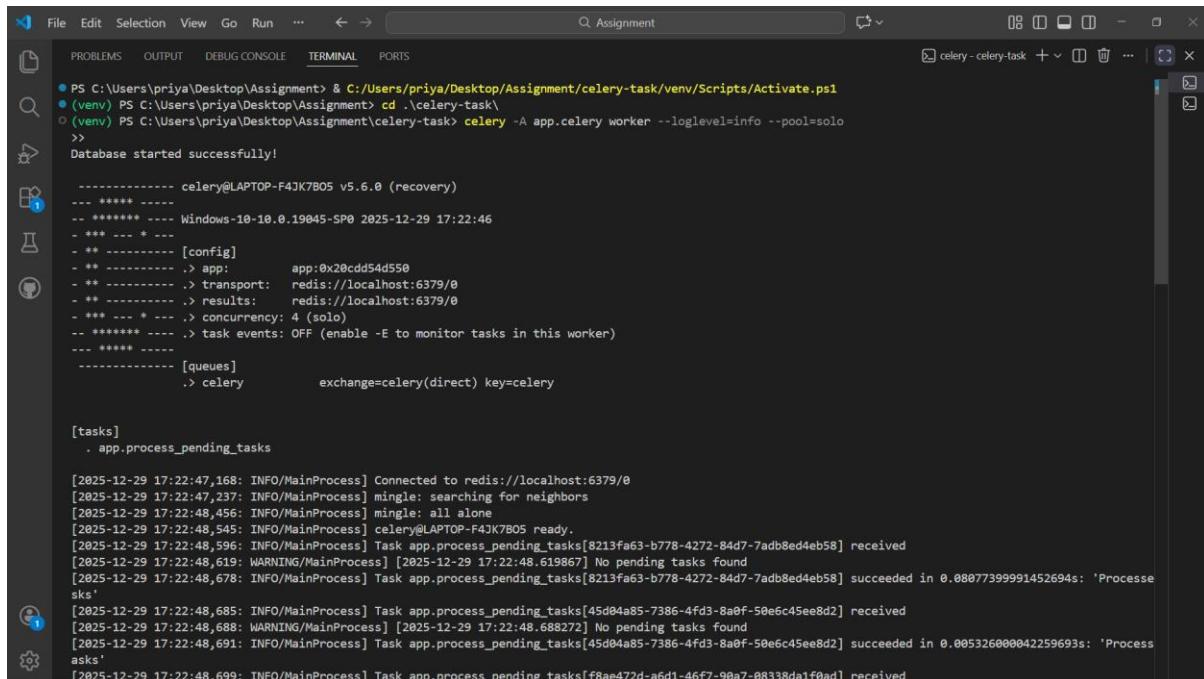


The screenshot shows the VS Code interface with the terminal tab active. The terminal window displays the following command-line session:

```
PS C:\Users\priya\Desktop\Assignment> & C:/Users/priya/Desktop/Assignment/celery-task/venv/Scripts/Activate.ps1
● (venv) PS C:\Users\priya\Desktop\Assignment> cd ..celery-task
○ (venv) PS C:\Users\priya\Desktop\Assignment\celery-task> python app.py
Database started successfully!
  * Serving Flask app 'app'
  * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
Press CTRL+C to quit
  * Restarting with stat
Database started successfully!
  * Debugger is active!
  * Debugger PIN: 881-253-683
```

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

9.Starting celery worker



The screenshot shows the VS Code interface with the terminal tab active. The terminal window displays the following command-line session:

```
PS C:\Users\priya\Desktop\Assignment> & C:/Users/priya/Desktop/Assignment/celery-task/venv/Scripts/Activate.ps1
● (venv) PS C:\Users\priya\Desktop\Assignment> cd ..celery-task
○ (venv) PS C:\Users\priya\Desktop\Assignment\celery-task> celery -A app.celery worker --loglevel=info --pool=solo
>>
Database started successfully!

----- celery@LAPTOP-F4JK7B05 v5.6.0 (recovery)
--- **** --- Windows-10-10.0.19045-SP0 2025-12-29 17:22:46
- ** --- *
- ** ----- [config]
- ** ----- > app:          app:0x20cd54d550
- ** ----- > transport:    redis://localhost:6379/0
- ** ----- > results:      redis://localhost:6379/0
- *** --- * --- > concurrency: 4 (solo)
-- ***** --- > task events: OFF (enable -E to monitor tasks in this worker)
--- ***** ----

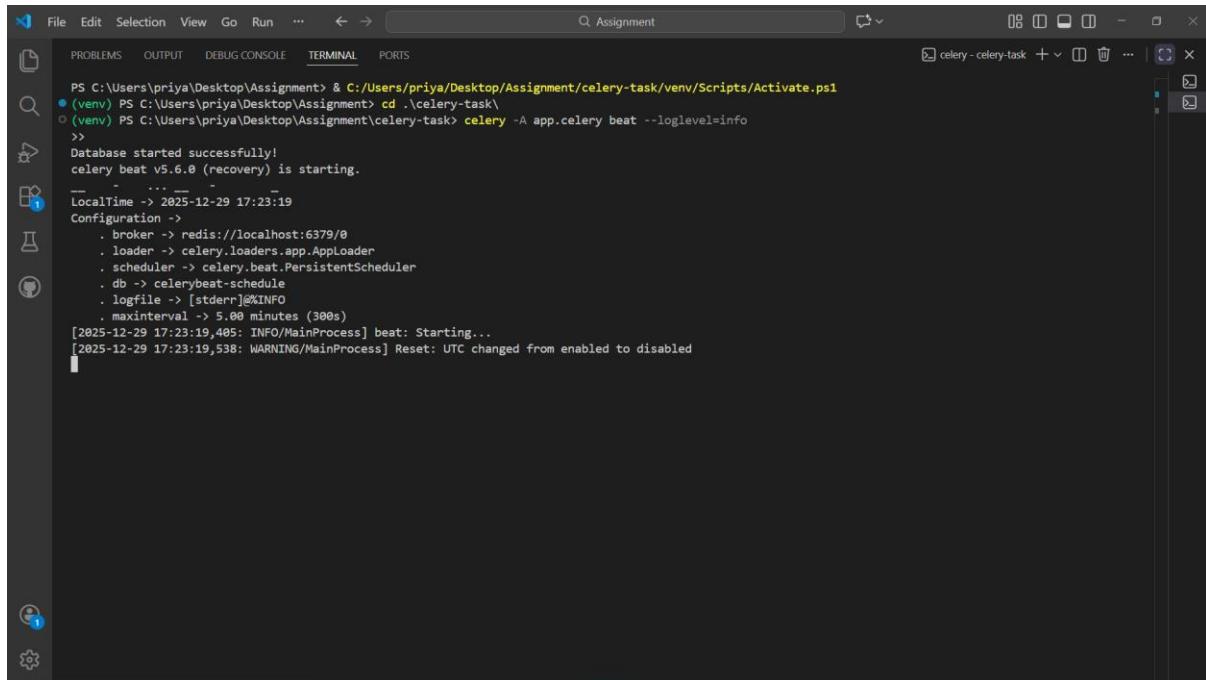
----- [queues]
     .> celery           exchange=celery(direct) key=celery

[tasks]
  . app.process_pending_tasks

[2025-12-29 17:22:47,168: INFO/MainProcess] Connected to redis://localhost:6379/0
[2025-12-29 17:22:47,237: INFO/MainProcess] mingle: searching for neighbors
[2025-12-29 17:22:48,456: INFO/MainProcess] mingle: all alone
[2025-12-29 17:22:48,545: INFO/MainProcess] celery@LAPTOP-F4JK7B05 ready.
[2025-12-29 17:22:48,596: INFO/MainProcess] Task app.process_pending_tasks[8213fa63-b778-4272-84d7-7adb8ed4eb58] received
[2025-12-29 17:22:48,619: WARNING/MainProcess] [2025-12-29 17:22:48.619867] No pending tasks found
[2025-12-29 17:22:48,678: INFO/MainProcess] Task app.process_pending_tasks[8213fa63-b778-4272-84d7-7adb8ed4eb58] succeeded in 0.08077399991452694s: 'Processes'
[2025-12-29 17:22:48,685: INFO/MainProcess] Task app.process_pending_tasks[45d04a85-7386-4fd3-8a0f-50e6c45ee8d2] received
[2025-12-29 17:22:48,688: WARNING/MainProcess] [2025-12-29 17:22:48.688272] No pending tasks found
[2025-12-29 17:22:48,691: INFO/MainProcess] Task app.process_pending_tasks[45d04a85-7386-4fd3-8a0f-50e6c45ee8d2] succeeded in 0.005326000042259693s: 'Processes'
[2025-12-29 17:22:48,699: INFO/MainProcess] Task app.process_pending_tasks[f8ae472d-a6d1-46f7-90a7-08338dal1f0ad] received
```

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

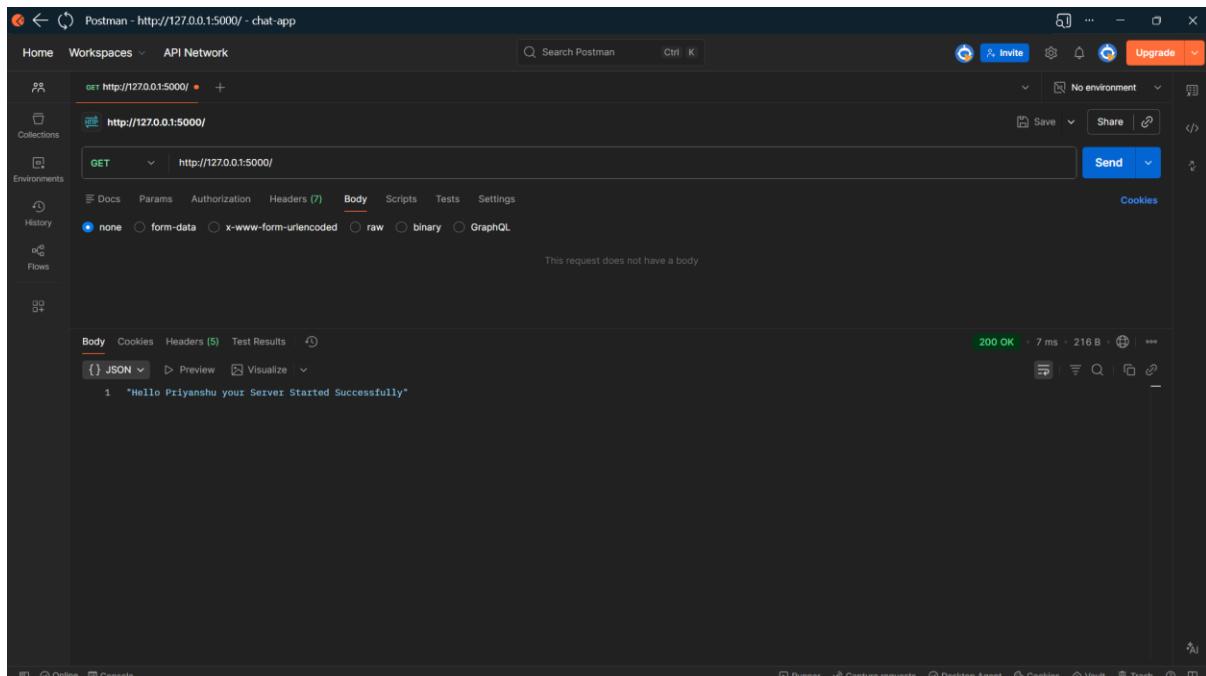
10. Celery Beat



```
PS C:\Users\priya\Desktop\Assignment> & C:/Users/priya/Desktop/Assignment/celery-task/venv/Scripts/Activate.ps1
● (venv) PS C:\Users\priya\Desktop\Assignment> cd .\celery-task\
○ (venv) PS C:\Users\priya\Desktop\Assignment\celery-task> celery -A app.celery beat --loglevel=info
>>
Database started successfully!
celery beat v5.6.0 (recovery) is starting.
-- 
LocalTime -> 2025-12-29 17:23:19
Configuration -
    . broker -> redis://localhost:6379/0
    . loader -> celery.loaders.app.AppLoader
    . scheduler -> celery.beat.PersistentScheduler
    . db -> celerybeat-schedule
    . logfile -> [stderr]@%INFO
    . maxinterval -> 5.00 minutes (300s)
[2025-12-29 17:23:19,405: INFO/MainProcess] beat: Starting...
[2025-12-29 17:23:19,538: WARNING/MainProcess] Reset: UTC changed from enabled to disabled
```

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

11. Starting the app on postman



The screenshot shows the Postman application interface. A GET request is made to `http://127.0.0.1:5000/`. The response status is 200 OK, with a response time of 7 ms and a body size of 216 B. The response body contains the JSON object: `{ "Hello Priyanshu your Server Started Successfully" }`.

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

12.Adding task to the DB

The screenshot shows the Postman interface. A POST request is made to `http://127.0.0.1:5000/add-task`. The request body contains a JSON object with a single key-value pair: `{"title": "first-task"}`. The response status is `200 OK`, and the response body is `{"success": true, "task_id": 4}`.

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

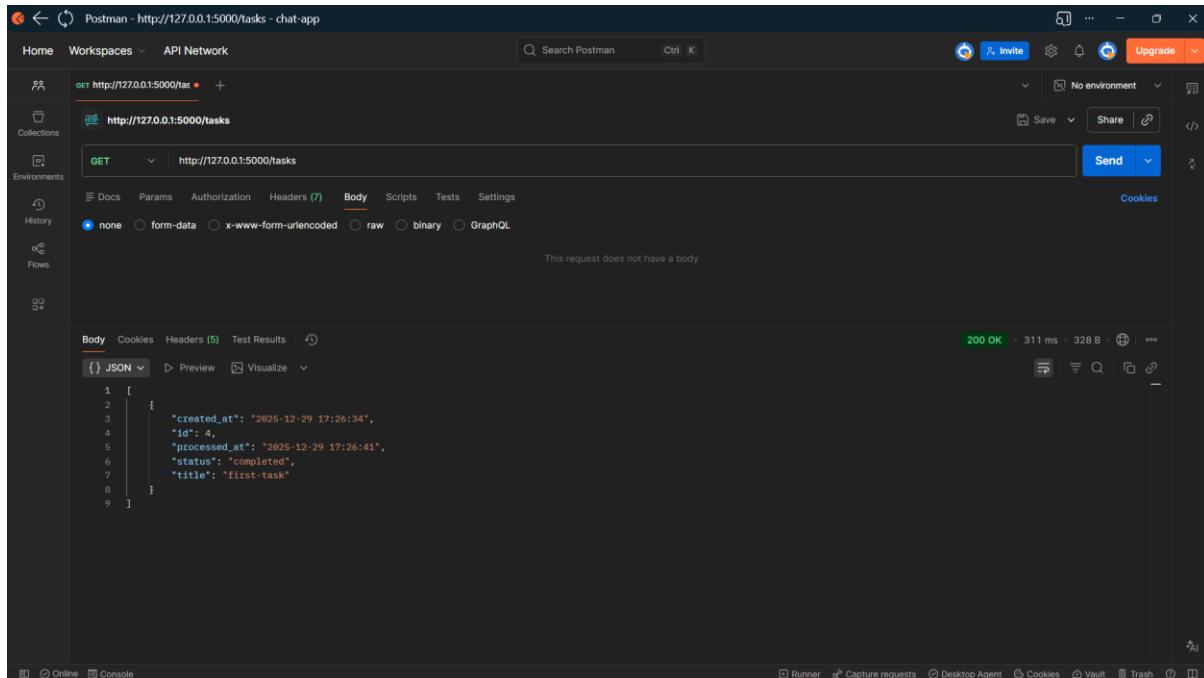
13.DB check at php-myadmin (XAMPP software)

The screenshot shows the phpMyAdmin interface connected to a MySQL database named `celery_task_manager`. The `tasks` table is selected. The table contains the following data:

	<code>id</code>	<code>title</code>	<code>status</code>	<code>created_at</code>	<code>processed_at</code>
	4	first-task	completed	2025-12-29 17:26:34	2025-12-29 17:26:41
	5	second-task	completed	2025-12-29 17:28:17	2025-12-29 17:28:21
	6	third-task	completed	2025-12-29 17:29:35	2025-12-29 17:29:41

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

14.Check for tasks



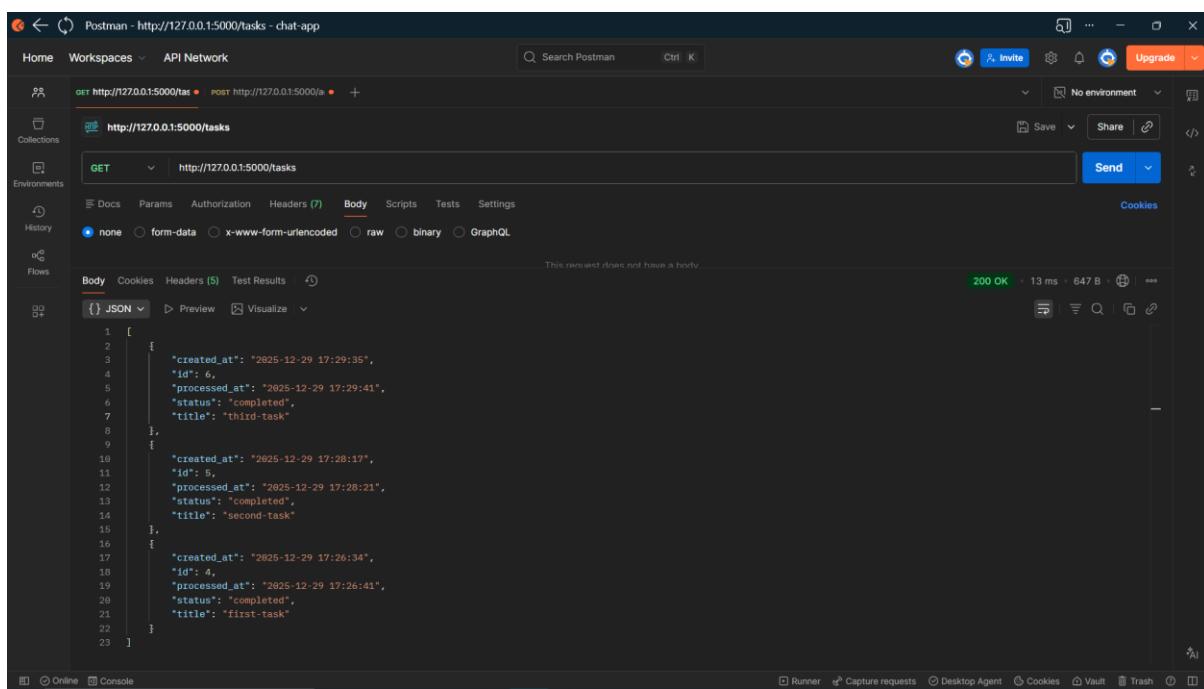
Postman - http://127.0.0.1:5000/tasks - chat-app

GET http://127.0.0.1:5000/tasks

Body

```
[{"id": 4, "title": "first-task", "status": "completed", "created_at": "2025-12-29 17:26:34", "processed_at": "2025-12-29 17:26:41"}]
```

200 OK



Postman - http://127.0.0.1:5000/tasks - chat-app

GET http://127.0.0.1:5000/tasks

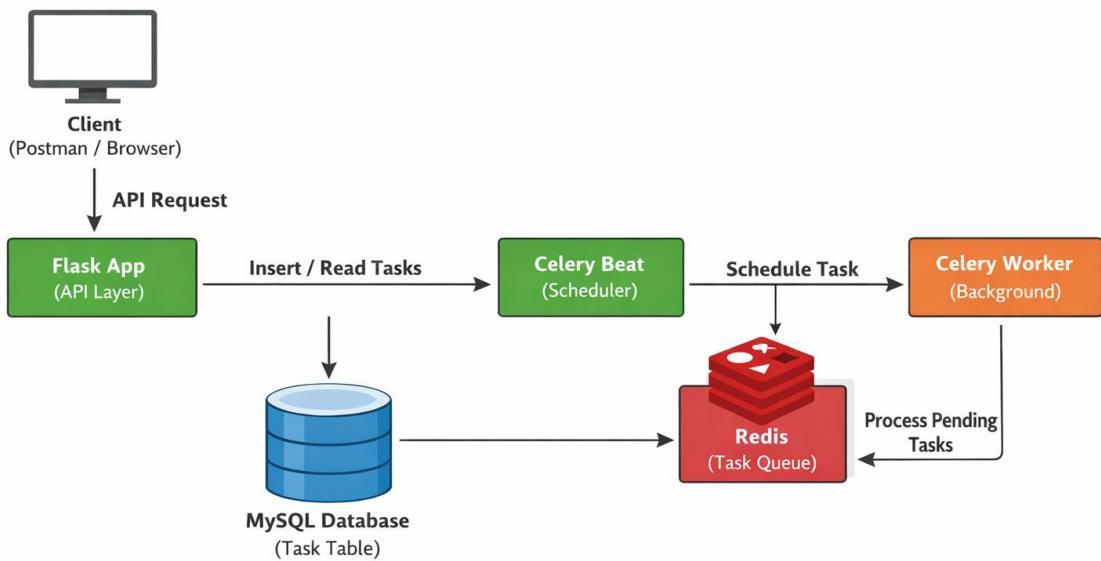
Body

```
[{"id": 6, "title": "third-task", "status": "completed", "created_at": "2025-12-29 17:29:35", "processed_at": "2025-12-29 17:29:41"}, {"id": 5, "title": "second-task", "status": "completed", "created_at": "2025-12-29 17:28:17", "processed_at": "2025-12-29 17:28:21"}, {"id": 4, "title": "first-task", "status": "completed", "created_at": "2025-12-29 17:26:34", "processed_at": "2025-12-29 17:26:41"}]
```

200 OK

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