

Documentation

Overall Purpose of the Project

To build a full-stack task execution system that:

- Allows users to **create, manage, and run tasks** (defined by shell commands).
- **Tracks** each task's execution history (start time, end time, output).
- Provides a **user-friendly frontend** to interact with tasks.
- Uses a **MongoDB backend** to store task definitions and execution logs.
- Automates command-line operations via a web interface, enabling technical users (like DevOps or engineers) to **execute scripts without needing terminal access**.

Backend (Spring Boot + MongoDB)

- REST API to handle CRUD operations for tasks.
- Executes shell commands when triggered.
- Stores the output, start/end time in MongoDB.
- Handles CORS to support frontend requests.
- Uses TaskController, Task, and TaskExecution models for logical separation.

Real-World Use Cases

- **DevOps Automation:** Run deployment, backup, or cleanup scripts from a web interface.
- **Admin Tooling:** Give operations teams the ability to execute predefined tasks without needing direct terminal access.
- **Remote Execution Dashboard:** Useful in cases where tasks must be managed or run remotely (e.g., servers, CI/CD scripts).
- **Education or Demonstration:** Useful for teaching how command-line tasks can be integrated into modern web systems.

Task-1

Backend Configuration

Task Management Backend – Overview

Purpose

The backend provides a RESTful API for managing shell-based tasks. It allows users to:

- Create and store task definitions
- Execute shell commands via tasks
- Track execution history
- Retrieve, search, and delete tasks

Architecture

- **Framework:** Spring Boot (Java)
- **Type:** RESTful Web Service
- **Data Access:** Spring Data JPA
- **Database:** MongoDB
- **Execution Logic:** Shell command execution tracked with timestamps

Core Components

1. Model: Task

Represents a task with:

- `id (String)`: Unique identifier
- `name (String)`: Task name
- `owner (String)`: Creator
- `command (String)`: Shell command to be executed
- `taskExecutions (List)`: History of runs

2. Model: TaskExecution

Represents a single execution of a task:

- id: UUID
- startTime: Execution start timestamp
- endTime: Execution end timestamp
- output: Output of the shell command

3. Controller: TaskController

Handles REST API endpoints:

- GET /tasks – List all tasks
- GET /tasks/{id} – Get task by ID
- GET /tasks/search?name= – Search by name
- PUT /tasks – Create/update task
- DELETE /tasks/{id} – Delete task
- PUT /tasks/{id}/execute – Run task and log execution

4. Repository: TaskRepository

Handles data persistence using JPA:

- Standard CRUD methods
- Custom search by name using:
List<Task> findByNameContainingIgnoreCase(String name);








CORS Handling

- @CrossOrigin(origins = "http://localhost:3000") allows requests from the React frontend running on a different port.

Execution Workflow

1. **User creates a task** with command and metadata.
2. **Task is saved** to the database.
3. On execution:
 - Backend runs the shell command using Java.
 - Captures start time, end time, and output.
 - Saves a new `TaskExecution` entry under the task.

Features Summary

Feature	Status
Create Task	
Search Task by Name	
Delete Task	
Execute Task Command	
Track Execution History	
REST API w/ JSON	
CORS Support	

Example JSON Response

```
{
  "id": "task01",
  "name": "Backup",
  "owner": "Admin",
  "command": "tar -czf backup.tar.gz /home/data",
  "taskExecutions": [
    {
      "startTime": "2025-06-11T13:30:00",
      "endTime": "2025-06-11T13:30:05",
      "output": "Backup completed"
    }
  ]
}
```

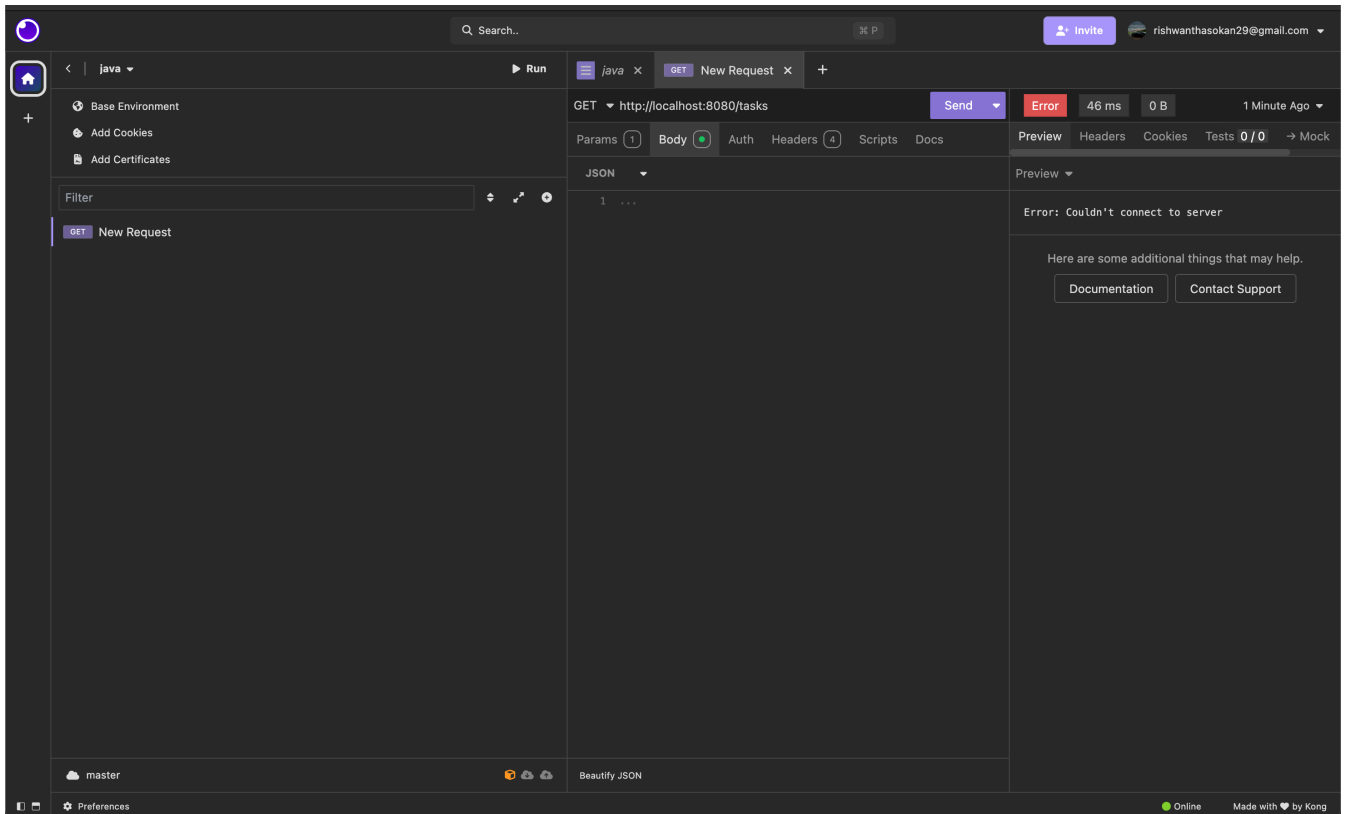
```
]
}
```

Api test Images - Insomnia

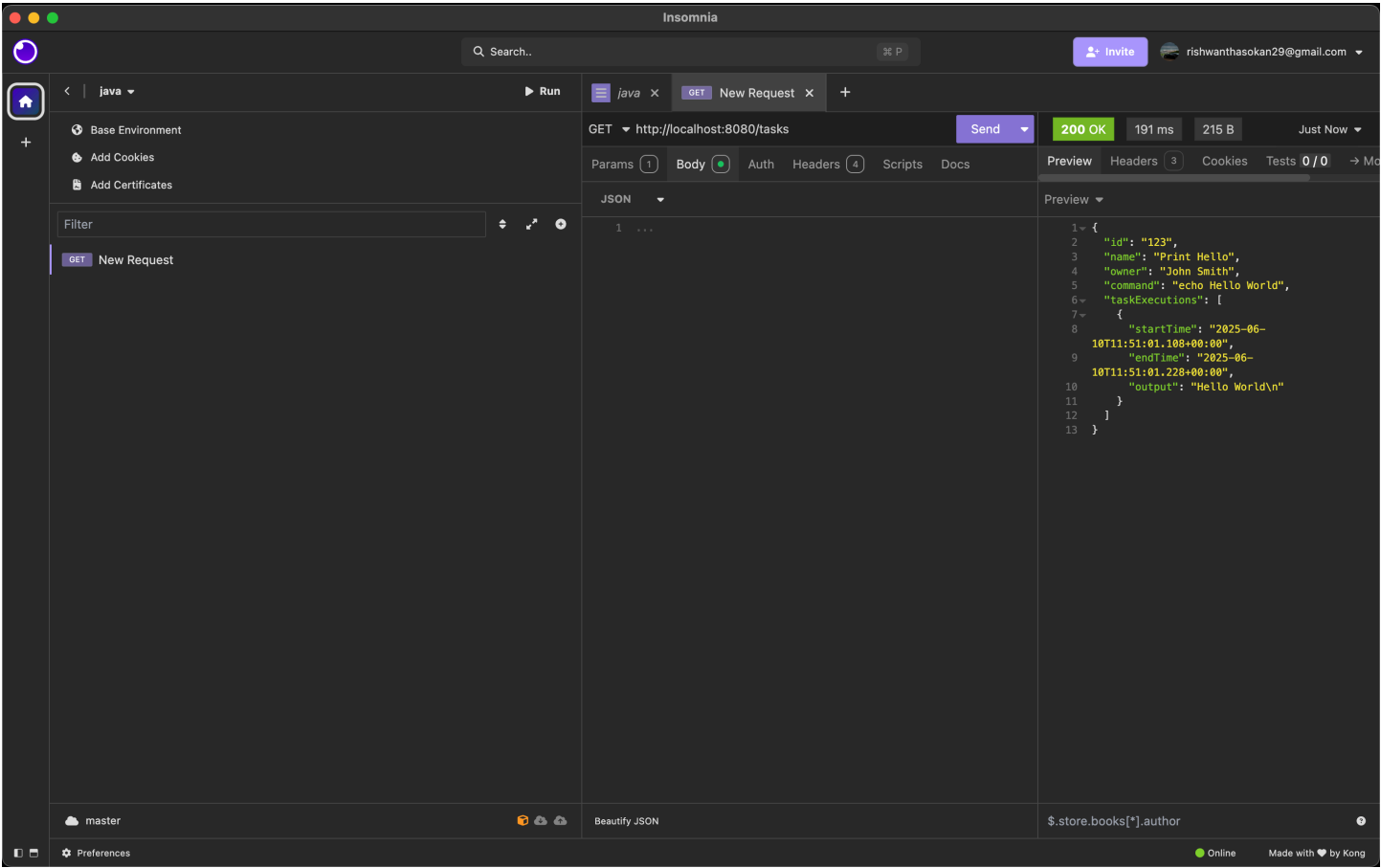
1, Api: <http://localhost:8080/tasks>

Method: Get

Description: Before Clicking Send Button



Description: After Clicking Send Button



2, **Api:** <http://localhost:8080/tasks>
Method: Put
Description: Creating new Data

The screenshot displays the Kong API Gateway web interface. The top navigation bar includes a search bar, a user profile icon, and the email address 'rishwanthasokan29@gmail.com'. The main interface is divided into three panels. The left panel shows the 'Base Environment' section with options to 'Add Cookies' and 'Add Certificates', and a 'Filter' input field. The middle panel shows the 'PUT' method selected for the endpoint 'http://localhost:8080/tasks'. The 'Body' tab is active, displaying a JSON payload:

```
1 {
2   "id": "125",
3   "name": "Print Hi",
4   "owner": "Rishwanth",
5   "command": "Hi World"
6 }
7
```

. The right panel shows the response status '200 OK' with a response time of '77 ms' and a size of '91 B'. The 'Preview' tab is active, displaying the response body:

```
1 {
2   "id": "125",
3   "name": "Print Hi",
4   "owner": "Rishwanth",
5   "command": "Hi World",
6   "taskExecutions": []
7 }
```

. The bottom status bar indicates the request was made from the 'master' environment and the response was beautified using 'Beautify JSON'.

3, **Api:** <http://localhost:8080/tasks?id=123>

Method: Get

Description: Getting task by id

The screenshot shows the Postman application interface. On the left, there's a sidebar with a 'Run' button and a 'GET New Request' button. The main area displays the request details for a GET request to `http://localhost:8080/tasks?id=123`. The response status is '200 OK' with a response time of '31 ms' and a size of '215 B'. The response body is a JSON object, which is displayed in the 'Preview' tab on the right. The JSON object contains the following data:

```
1 {
2   "id": "123",
3   "name": "Print Hello",
4   "owner": "John Smith",
5   "command": "echo Hello World",
6   "taskExecutions": [
7     {
8       "startTime": "2025-06-
9       10T11:51:01.108+00:00",
10      "endTime": "2025-06-
11      10T11:51:01.228+00:00",
12      "output": "Hello World\n"
13    }
14  ]
15 }
```


4, **Api:** <http://localhost:8080/tasks?id=126>

Method: Get

Description: Getting task by id, if no task found through error message

