

Word Finder Assignment Report

Name: Rahul Kumar Sahu

Email: rahulsahu1021@gmail.com

Contact: +917016806036

Problem: Create a REST API that takes in multiple paragraphs of text as input, stores each paragraph and the words to paragraph mappings on a PostgreSQL database.

Software Stack:

- Server – Node.js, Express.js
 - Database – PostgreSQL
 - Dependencies – pg, nodemon, jsonwebtoken, dotenv, bcrypt
-

NOTE: This assignment exclusively contains backend functionality as the frontend was not requested. Additionally, this assignment is not based on Django Rest Framework and does not use Docker and Docker-compose since I am not yet familiar with them.

Process:

- 1) Setting up Environment and Tools:
 - a. Installed Node.js and npm (Node Package Manager).
 - b. Installed PostgreSQL and set up a database.
 - c. Initialized Node.js Project:
- 2) Creating a new directory for the project:
 - a. Run npm init to initialize a new Node.js project.
 - b. Install necessary dependencies like Express, pg (PostgreSQL client), jsonwebtoken (for JWT authentication), etc.
 - c. Create PostgreSQL Database: 'WordFinder'
- 3) Using PostgreSQL to create database tables:
 - a. Create tables for user data and paragraph data and word mapping:
 - i. Users
 - ii. Paragraphs
 - iii. Word_paragraph_mapping
- 4) Connect database with the server
- 5) Setting up REST APIs methods:
 - a. Routes for users-signup, login, logout
 - b. Routes for paragraphs – CRUD operations,
 - c. Route for multiple paragraphs in single input
 - d. Route for searching word
- 6) Middleware Implementation:

- a. Middleware implementation using JWT for user authentication
- b. Middleware implementation for handling server errors

7) Controllers for each routes.

8) Storing Multiple Paragraphs from single input functionality :

the createParagraphBatch function receives a batch of paragraphs, inserts them into a database, maps each word within those paragraphs to another table, and then sends a success response. It handles any errors that occur during the process.

The splitting of paragraphs is handled by the splitIntoParagraphs function, which takes a block of text as input and splits it into individual paragraphs. It does this by using the `\n\n` delimiter, which represents two consecutive line breaks in the text. Each paragraph is then trimmed to remove any leading or trailing whitespace. This process ensures that each paragraph is extracted accurately from the input text.

9) Mapping words in database functionality:

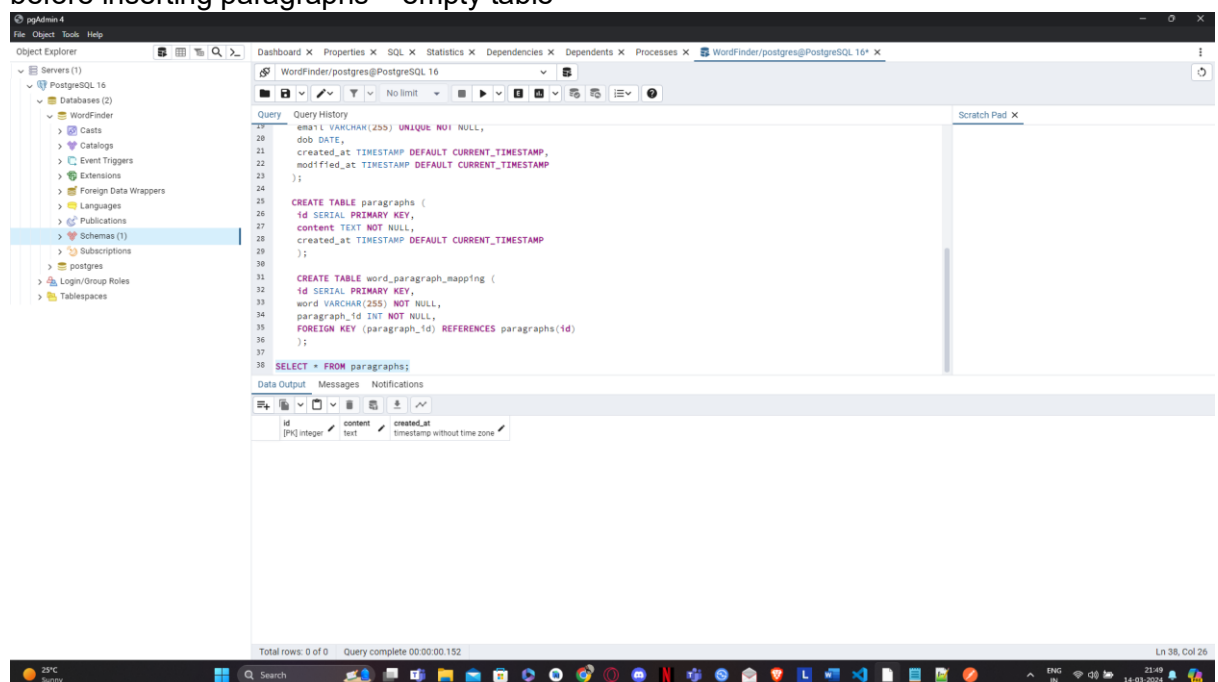
he function mapWordsToParagraphs retrieves all paragraphs from the database and iterates over each paragraph. For each paragraph, it splits the content into individual words using a regular expression `\s+` to match one or more whitespace characters. Then, it iterates over each word and inserts it along with the paragraph ID into the `word_paragraph_mapping` table in the database.

10) Testing all the endpoints using POSTMAN

Demo:

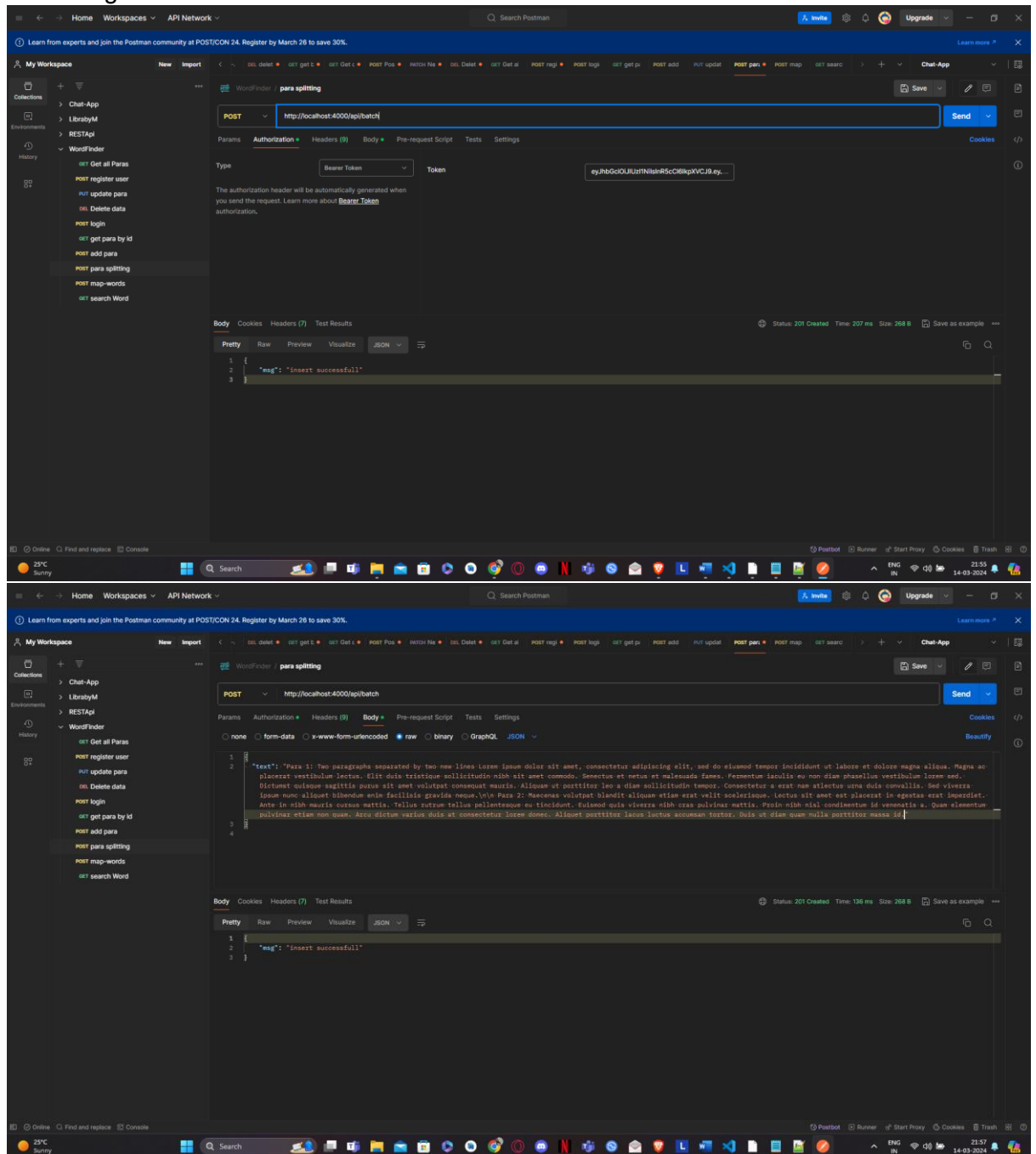
1) Adding Paragraphs:

before inserting paragraphs – empty table

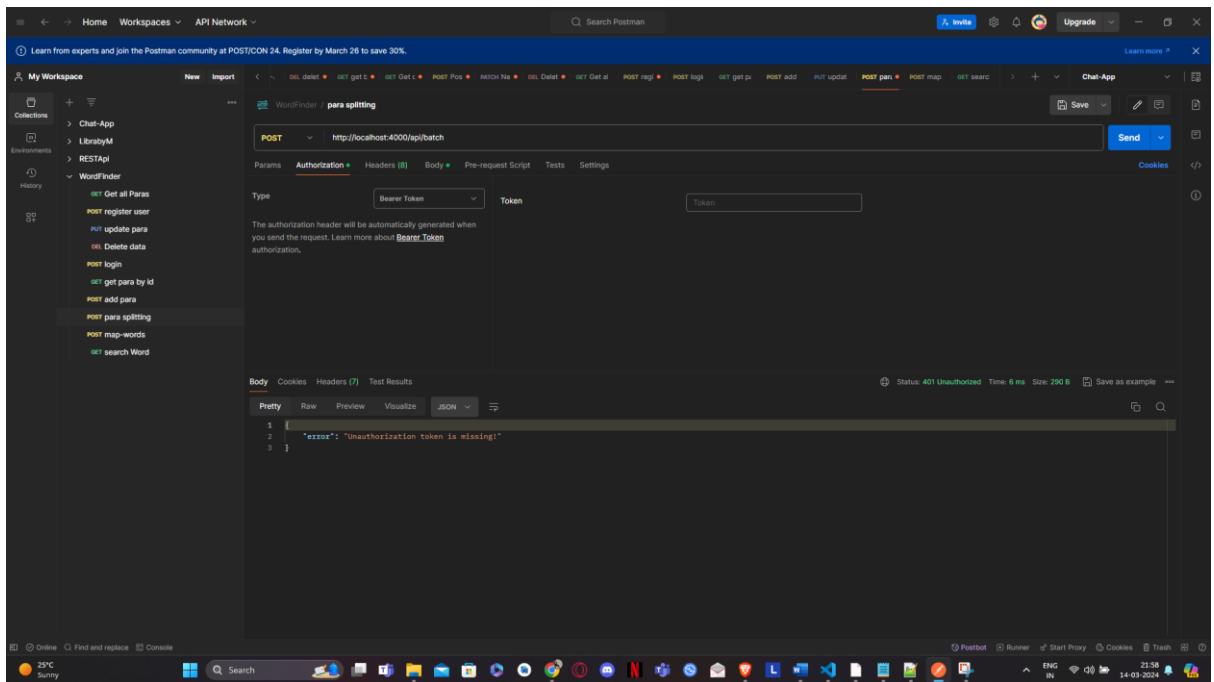


inserting paragraphs through endpoints using postman

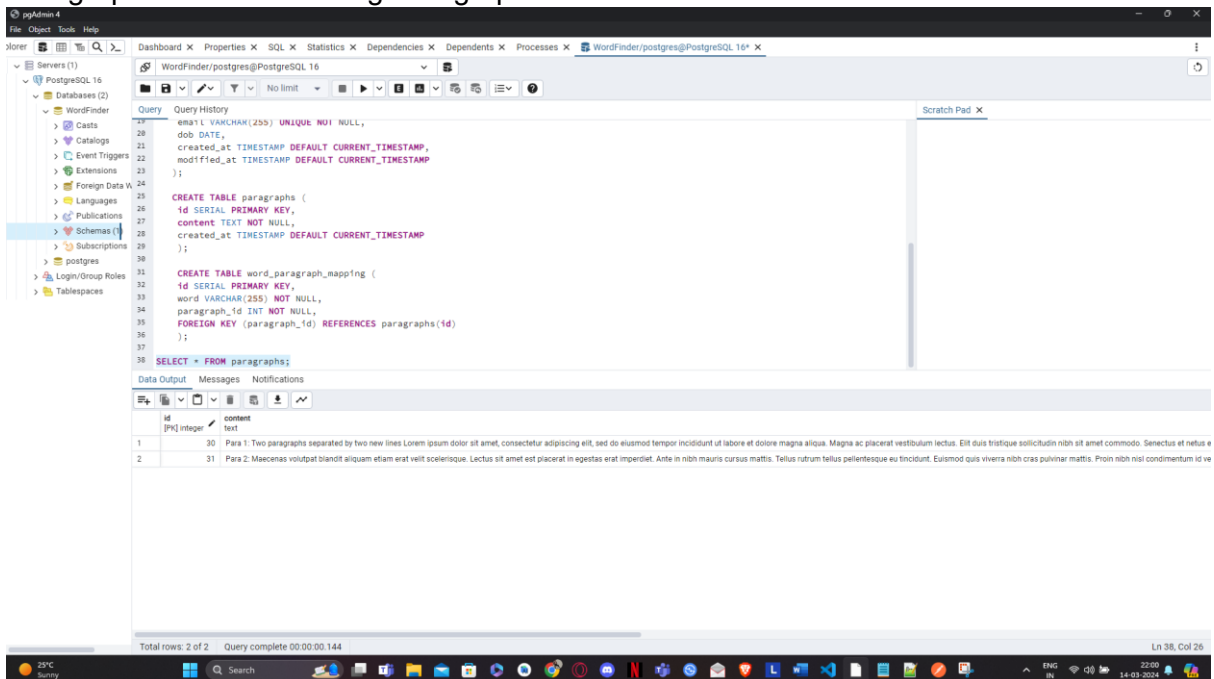
with using bearer token:



Using without bearer token:



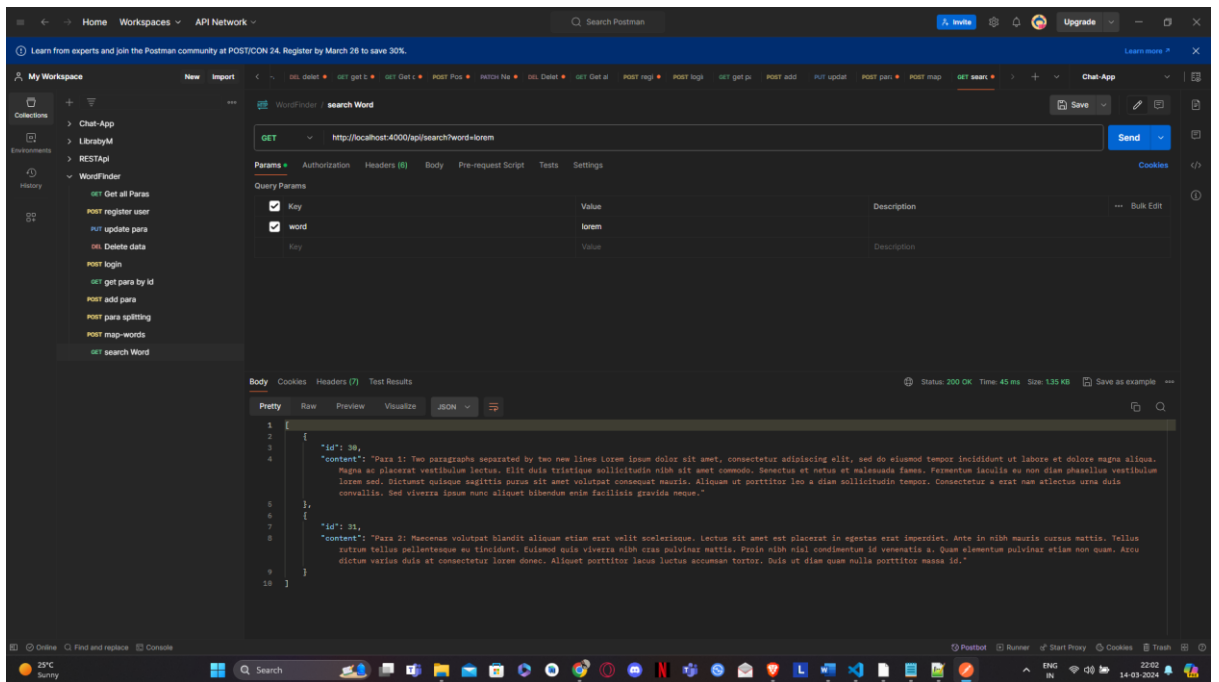
Paragraph Table after adding Paragraphs:



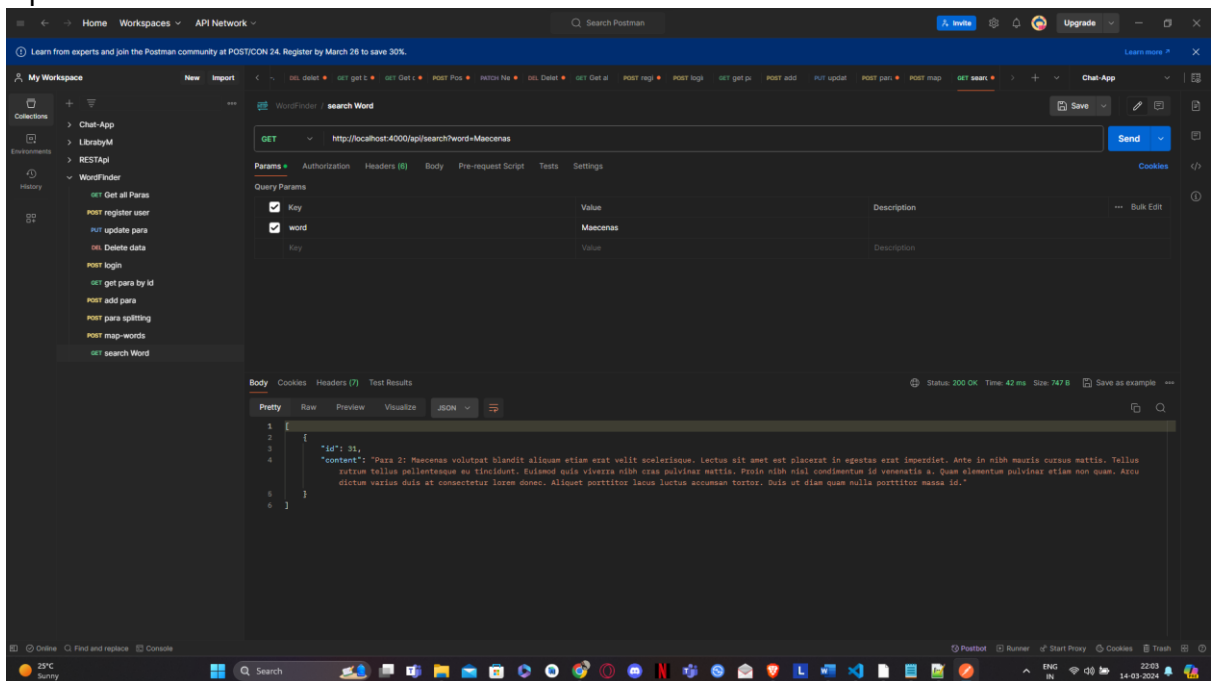
2) Searching Words:

Using Postman to take query parameter to return Paragraphs:

input: lorem:



input: - Maecenas:



Tables:
Paragraph table:

pgAdmin 4

Dashboard | Properties | SQL | Statistics | Dependencies | Dependents | Processes | WordFinder/postgres@PostgreSQL 16*

Query History

```
20 CREATE TABLE paragraphs (
21   id SERIAL PRIMARY KEY,
22   content TEXT NOT NULL,
23   created_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP,
24   modified_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP
25 );
26
27 CREATE TABLE word_paragraph_mapping (
28   id SERIAL PRIMARY KEY,
29   word VARCHAR(255) NOT NULL,
30   paragraph_id INT NOT NULL,
31   FOREIGN KEY (paragraph_id) REFERENCES paragraphs(id)
32 );
33
34 SELECT * FROM paragraphs;
```

Data Output

id	content
1	Para 1: Two paragraphs separated by two new lines Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Magna ac placerat vestibulum lectus. Eit dui tristique sollicitudin nibh sit amet commodo. Senectus et netus e
2	Para 2: Maecenas volutpat blandit aliquam elit et velit scelerisque. Lectus sit amet est placerat in aegestas erat imperdiet. Ante in nibh mauris cursus mattis. Tellus rutrum tellus pellentesque eu tincidunt. Euismod quis viverra nibh cras pulvinar mattis. Proin nibh nisl condimentum id ve

Total rows: 2 of 2 Query complete 00:00:00.144

Word_paragraph_mapping table:

pgAdmin 4

Dashboard | Properties | SQL | Statistics | Dependencies | Dependents | Processes | WordFinder/postgres@PostgreSQL 16*

Query History

```
30 CREATE TABLE word_paragraph_mapping (
31   id SERIAL PRIMARY KEY,
32   word VARCHAR(255) NOT NULL,
33   paragraph_id INT NOT NULL,
34   FOREIGN KEY (paragraph_id) REFERENCES paragraphs(id)
35 );
36
37 SELECT * FROM paragraphs;
38 SELECT * FROM word_paragraph_mapping;
```

Data Output

id	word	paragraph_id
1	Para	30
2	1:	30
3	Two	30
4	paragraphs	30
5	separated	30
6	by	30
7	two	30
8	new	30
9	lines	30
10	Lorem	30
11	ipsum	30
12	dolor	30
13	sit	30
14	amet	30
15	consectetur	30
16	adipiscing	30
17	elit	30
18	sed	30
19	do	30
20	eiusmod	30

Total rows: 164 of 164 Query complete 00:00:00.148

Users table:

