

Name - Yash Lakhtariya
Enrollment number - 21162101012
Branch - CBA Batch - 61
EADC Practical 19

Aim : Implement an application with CRUD operation using Postgres with NodeJS.

A social media website xyz.com wants to manage its users records and this task is given to you. They are seeking functionalities :

Practical 19.1: New joiner data should be stored in DB.

Practical 19.2: The admin can see the whole DB

Practical 19.3: The admin can filter out the DB.

Practical 19.4: The user can update their information, which should be updated in the DB.

Practical 19.5: The admin can delete the record from DB.

Practical 19.6: Integrate your Nodejs application with the IBM cloud service- Databases for Postgresql and check for the database status using pg admin.

Name - Yash Lakhtariya
Enrollment number - 21162101012
Branch - CBA Batch - 61
EADC Practical 19

Steps and Screenshots:

1. Install and setup postgresql and pgadmin in local machine and try creating database, table and insert values in it

The screenshot shows a terminal window titled "pgcli -U postgres ~". The session starts with the PostgreSQL server information: "Server: PostgreSQL 16.2", "Version: 4.0.1", and "Home: http://pgcli.com". The user then creates a database named "api" with the command "CREATE DATABASE api;". After a short delay, the user creates a table named "COMPANY" with the following schema:
CREATE TABLE COMPANY(ID INT PRIMARY KEY
NOT NULL, NAME TEXT NOT NULL, AGE INT
NOT NULL, ADDRESS CHAR(50), SALARY REAL);
The user then creates another table named "TABLE" with the command "CREATE TABLE". Finally, the user inserts a value into the "COMPANY" table with the command "INSERT INTO COMPANY VALUES (1, "Hare Krishna", 20, "Kanpur", 50000);". The terminal window also displays system status at the bottom, including network traffic ("12:2556 PM ↑106B ↓304B") and system icons.

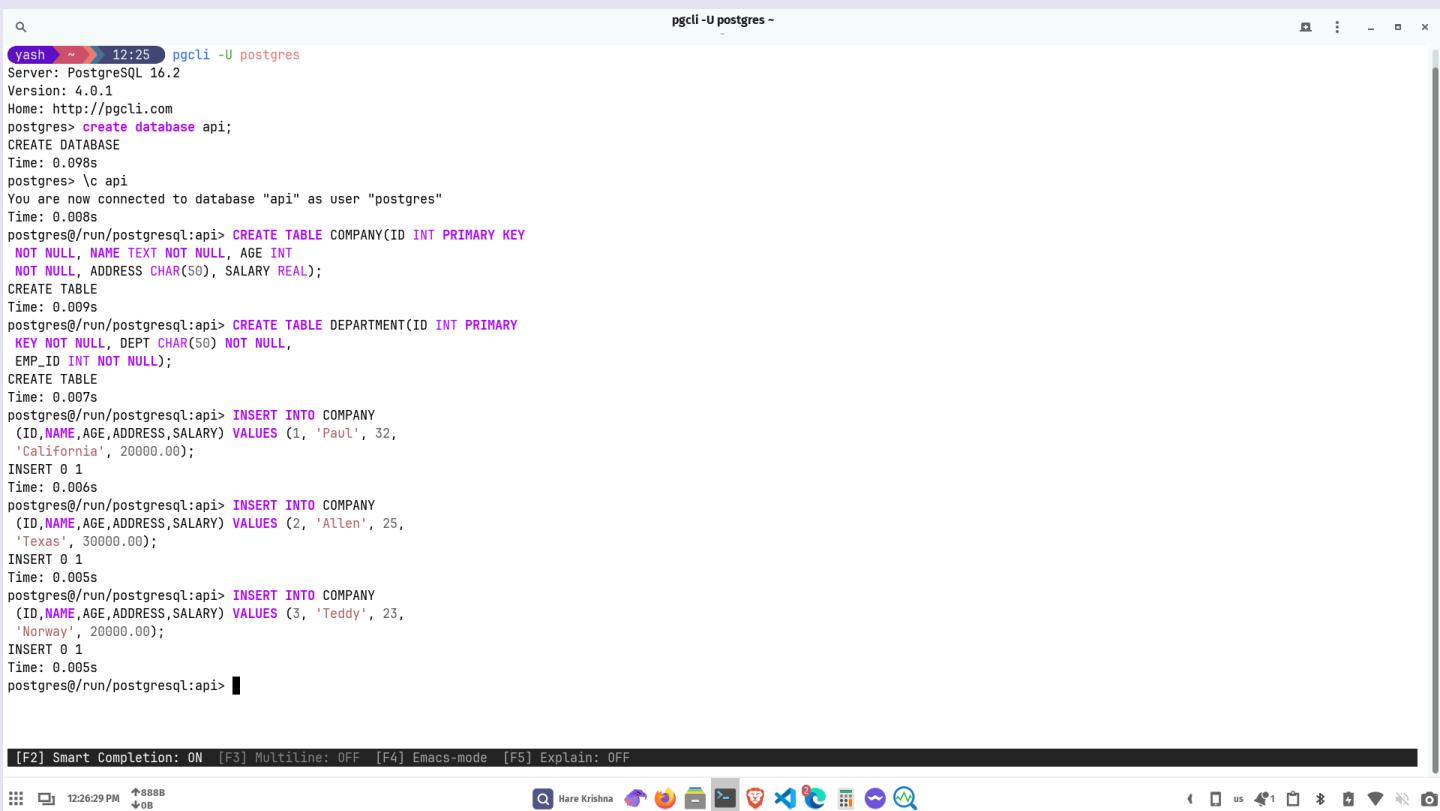
```
yash ~ 12:25 pgcli -U postgres
Server: PostgreSQL 16.2
Version: 4.0.1
Home: http://pgcli.com
postgres> create database api;
CREATE DATABASE
Time: 0.098s
postgres> \c api
You are now connected to database "api" as user "postgres"
Time: 0.008s
postgres@/run/postgresql:api> CREATE TABLE COMPANY(ID INT PRIMARY KEY
NOT NULL, NAME TEXT NOT NULL, AGE INT
NOT NULL, ADDRESS CHAR(50), SALARY REAL);
CREATE TABLE
Time: 0.009s
postgres@/run/postgresql:api> \i
[F2] Smart Completion: ON [F3] Multiline: OFF [F4] Emacs-mode [F5] Explain: OFF
[12:2556 PM ↑106B ↓304B] Hare Krishna
```

Name - Yash Lakhtariya
Enrollment number - 21162101012
Branch - CBA Batch - 61
EADC Practical 19

```
yash ~ 12:25 pgcli -U postgres
Server: PostgreSQL 16.2
Version: 4.0.1
Home: http://pgcli.com
postgres> create database api;
CREATE DATABASE
Time: 0.098s
postgres> \c api
You are now connected to database "api" as user "postgres"
Time: 0.088s
postgres@/run/postgresql:api> CREATE TABLE COMPANY(ID INT PRIMARY KEY
NOT NULL, NAME TEXT NOT NULL, AGE INT
NOT NULL, ADDRESS CHAR(50), SALARY REAL);
CREATE TABLE
Time: 0.009s
postgres@/run/postgresql:api> CREATE TABLE DEPARTMENT(ID INT PRIMARY
KEY NOT NULL, DEPT CHAR(50) NOT NULL,
EMP_ID INT NOT NULL);
CREATE TABLE
Time: 0.007s
postgres@/run/postgresql:api> INSERT INTO COMPANY
(ID,NAME,AGE,ADDRESS,SALARY) VALUES (1, 'Paul', 32,
'California', 20000.00);
INSERT 0 1
Time: 0.006s
postgres@/run/postgresql:api> INSERT INTO COMPANY
(ID,NAME,AGE,ADDRESS,SALARY) VALUES (2, 'Allen', 25,
'Texas', 30000.00);
INSERT 0 1
Time: 0.005s
postgres@/run/postgresql:api> INSERT INTO COMPANY
(ID,NAME,AGE,ADDRESS,SALARY) VALUES (3, 'Teddy', 23,
'Norway', 20000.00);
INSERT 0 1
Time: 0.005s
postgres@/run/postgresql:api>
```

[F2] Smart Completion: ON [F3] Multiline: OFF [F4] Emacs-mode [F5] Explain: OFF

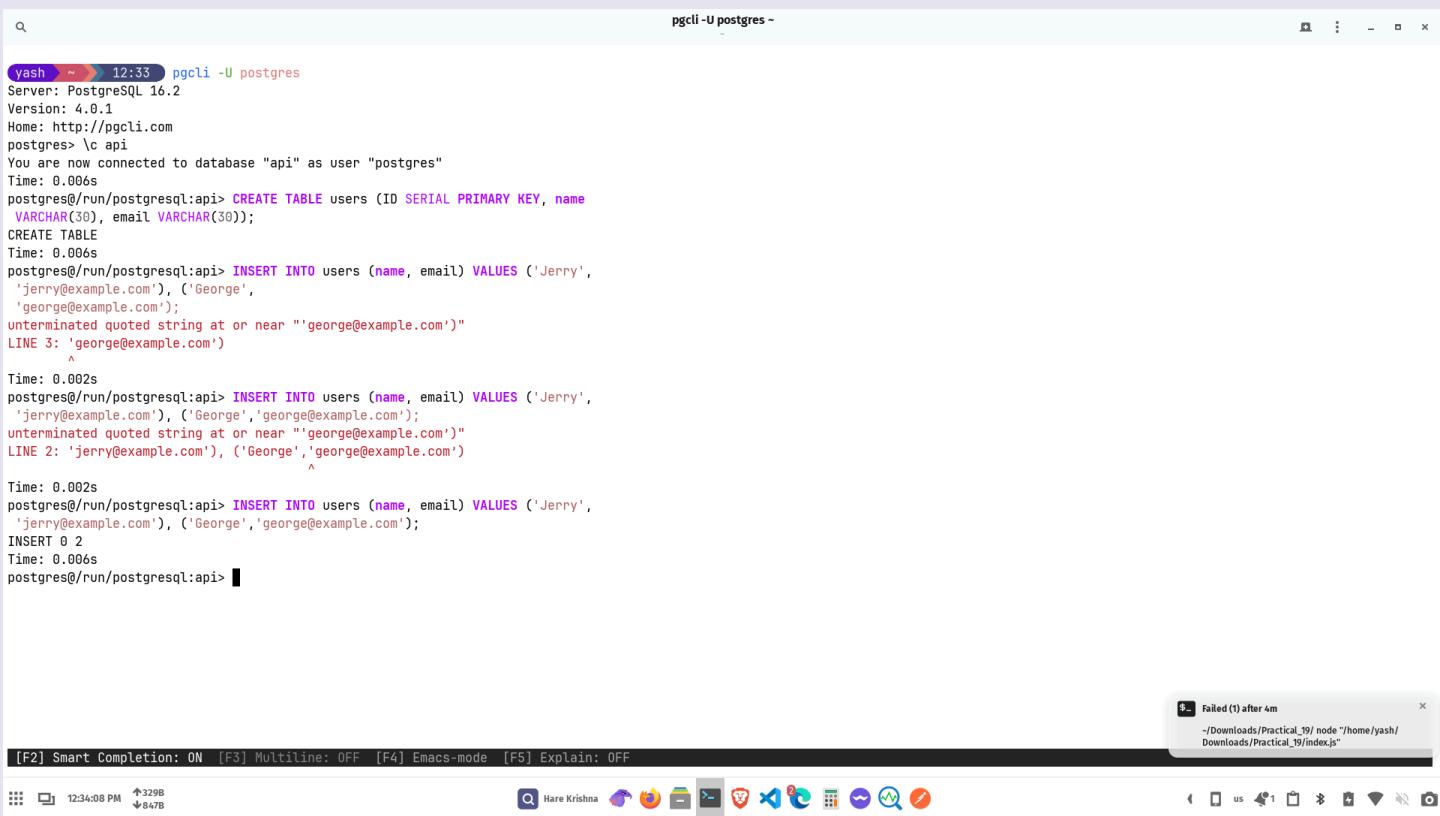
12:26:29 PM ↑8888 ↓08



```
yash ~ 12:33 pgcli -U postgres
Server: PostgreSQL 16.2
Version: 4.0.1
Home: http://pgcli.com
postgres> \c api
You are now connected to database "api" as user "postgres"
Time: 0.006s
postgres@/run/postgresql:api> CREATE TABLE users (ID SERIAL PRIMARY KEY, name
VARCHAR(30), email VARCHAR(30));
CREATE TABLE
Time: 0.006s
postgres@/run/postgresql:api> INSERT INTO users (name, email) VALUES ('Jerry',
'jerry@example.com'), ('George',
'george@example.com');
unterminated quoted string at or near "'george@example.com'"
LINE 3: 'george@example.com'
^
Time: 0.002s
postgres@/run/postgresql:api> INSERT INTO users (name, email) VALUES ('Jerry',
'jerry@example.com'), ('George', 'george@example.com');
unterminated quoted string at or near "'george@example.com'"
LINE 2: 'jerry@example.com', ('George', 'george@example.com')
^
Time: 0.002s
postgres@/run/postgresql:api> INSERT INTO users (name, email) VALUES ('Jerry',
'jerry@example.com'), ('George', 'george@example.com');
INSERT 0 2
Time: 0.006s
postgres@/run/postgresql:api>
```

[F2] Smart Completion: ON [F3] Multiline: OFF [F4] Emacs-mode [F5] Explain: OFF

12:34:08 PM ↑3298 ↓8478



Name - Yash Lakhtariya
Enrollment number - 21162101012
Branch - CBA Batch - 61
EADC Practical 19

2. Now run locally the node server to manage data in postgres database

The screenshot shows the Visual Studio Code interface with two open files: `index.js` and `queries.js`. The `index.js` file contains the main application logic using Express.js and a database connection. The `queries.js` file is imported into `index.js`. The terminal window shows the command `node index.js` being run, and the output indicates the application is listening on port 3000. To the right, a Microsoft Edge browser window displays the running application at `localhost:3000`, showing a JSON response with the message "Node.js, Express, and Postgres API".

Code (queries.js) :

```
const Pool = require('pg').Pool
var fs = require('fs');
const pool = new Pool({
  user: 'postgres',
  host: 'localhost',
  database: 'api',
  password: 'krsna',
  port: 5432,
  /*ssl: {
    rejectUnauthorized: false,
    cert: fs.readFileSync('./cert.pem').toString(),
  }
});
```

Name - Yash Lakhtariya
Enrollment number - 21162101012
Branch - CBA Batch - 61
EADC Practical 19

```
} */
})

const getUsers = (request, response) => {
  pool.query('SELECT * FROM users ORDER BY id ASC', (error, results) => {
    if (error) {
      throw error
    }
    response.status(200).json(results.rows)
  })
}

const getUserById = (request, response) => {
  const id = parseInt(request.params.id)
  pool.query('SELECT * FROM users WHERE id = $1', [id], (error, results) => {
    if (error) {
      throw error
    }
    response.status(200).json(results.rows)
  })
}

const createUser = (request, response) => {
  const { name, email } = request.body
  pool.query('INSERT INTO users (name, email) VALUES ($1, $2) RETURNING *', [name, email], (error, results) => {
    if (error) {
      throw error
    }
    response.status(201).send(`User added with ID: ${results.rows[0].id}`)
  })
}
```

Name - Yash Lakhtariya
Enrollment number - 21162101012
Branch - CBA Batch - 61
EADC Practical 19

```
    })
}

const updateUser = (request, response) => {
  const id = parseInt(request.params.id)
  const { name, email } = request.body
  pool.query(
    'UPDATE users SET name = $1, email = $2 WHERE id = $3',
    [name, email, id],
    (error, results) => {
      if (error) {
        throw error
      }
      response.status(200).send(`User modified with ID: ${id}`)
    }
  )
}

const deleteUser = (request, response) => {
  const id = parseInt(request.params.id)
  pool.query('DELETE FROM users WHERE id = $1', [id], (error, results)
=> {
  if (error) {
    throw error
  }
  response.status(200).send(`User deleted with ID: ${id}`)
})
}

module.exports = {
  getUsers,
  getUserById,
  createUser,
  updateUser,
  deleteUser,
```

Name - Yash Lakhtariya
Enrollment number - 21162101012
Branch - CBA Batch - 61
EADC Practical 19

}

Code (index.js) :

```
const express = require('express')
const bodyParser = require('body-parser')
const app = express()
const db = require('./queries')
const port = 3000

app.use(bodyParser.json())
app.use(
  bodyParser.urlencoded({
    extended: true,
  })
)

app.get('/', (request, response) => {
  response.json({ info: 'Node.js, Express, and Postgres API' })
})

app.get('/users', db.getUsers)
app.get('/users/:id', db.getUserById)
app.post('/users', db.createUser)
app.put('/users/:id', db.updateUser)
app.delete('/users/:id', db.deleteUser)

app.listen(port, () => {
  console.log(`App running on port ${port}.`)
})
```

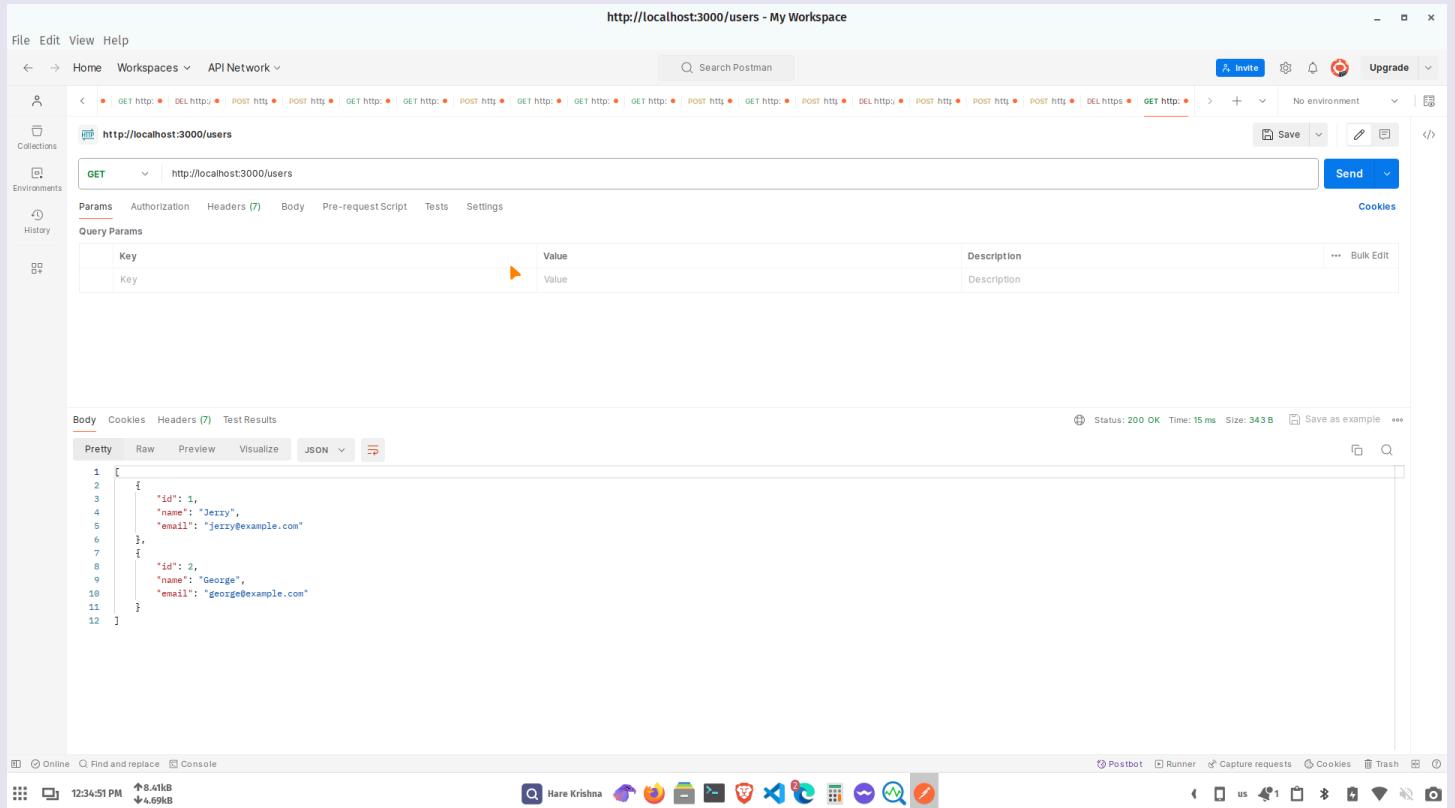
Name - Yash Lakhtariya
Enrollment number - 21162101012
Branch - CBA Batch - 61
EADC Practical 19

3. As the local server is running, use Postman to send requests for performing operations :

a) To get users table content in json format

Request : GET

URL : <http://localhost:port/users>



The screenshot shows the Postman application interface. The top bar displays the URL `http://localhost:3000/users`. The main workspace shows a GET request to `http://localhost:3000/users`. The 'Body' tab is selected, showing a JSON response with two user objects:

```
1 [ { "id": 1, "name": "Jerry", "email": "jerry@example.com" }, { "id": 2, "name": "George", "email": "george@example.com" } ]
```

The status bar at the bottom indicates a 200 OK status with a time of 15 ms and a size of 343 B.

Name - Yash Lakhtariya
Enrollment number - 21162101012
Branch - CBA Batch - 61
EADC Practical 19

b) Get details of specific user using ID

Request : GET

URL : <http://localhost:port/users/:id>

The screenshot shows the Postman application interface. The URL in the request field is `http://localhost:3000/users/1`. The response body is displayed in Pretty JSON format:

```
1 [  
2   {  
3     "id": 1,  
4     "name": "Jerry",  
5     "email": "jerry@example.com"  
6   }  
7 ]
```

The status bar at the bottom indicates a `200 OK` response with a time of `8 ms` and a size of `288 B`.

Name - Yash Lakhtariya
Enrollment number - 21162101012
Branch - CBA Batch - 61
EADC Practical 19

c) To add record of a user in users table

Request : POST

URL : <http://localhost:port/users>

Body : json object of user data

The screenshot shows the Postman application interface. The URL in the header is <http://localhost:3000/users>. A POST request is being made to <http://localhost:3000/users>. The request body is a JSON object:

```
1 {  
2   "id": 3,  
3   "name": "YSL",  
4   "email": "ysl@haribol.hari"  
5 }
```

The response status is 201 Created, time: 21ms, size: 254 B. The response body is: "User added with ID: 3".

Name - Yash Lakhtariya
Enrollment number - 21162101012
Branch - CBA Batch - 61
EADC Practical 19

d) Update the detail(s) of a user using ID

Request : PUT

URL : <http://localhost:port/users/:id>

Body : json object of updated user details

The screenshot shows the Postman application interface. The URL in the header is `http://localhost:3000/users/3 - My Workspace`. The request method is set to `PUT` and the endpoint is `http://localhost:3000/users/3`. The `Body` tab is selected, showing a JSON object:

```
1 {  
2   ... "id": 3,  
3   ... "name": "Yash",  
4   ... "email": "yel@haribol.hari"  
5 }
```

The response status is `200 OK`, time `11ms`, size `252 B`. The response body is: `User modified with ID: 3`.

Name - Yash Lakhtariya
Enrollment number - 21162101012
Branch - CBA Batch - 61
EADC Practical 19

e) Delete the record of a user using ID

Request : DELETE

URL : <http://localhost:port/users/:id>

The screenshot shows the Postman application interface. The URL in the header is <http://localhost:3000/users/3>. The request method is set to **DELETE** under the **Body** tab. The body contains the following JSON payload:

```
1 | ... {
2 | ...     "id": 3,
3 | ...     "name": "Yash",
4 | ...     "email": "yashharibolhari"
5 | ... }
```

The response status is **200 OK**, Time: 11ms, Size: 251B. The response body is: **User deleted with ID: 3**.

Name - Yash Lakhtariya

Enrollment number - 21162101012

Branch - CBA **Batch - 61**

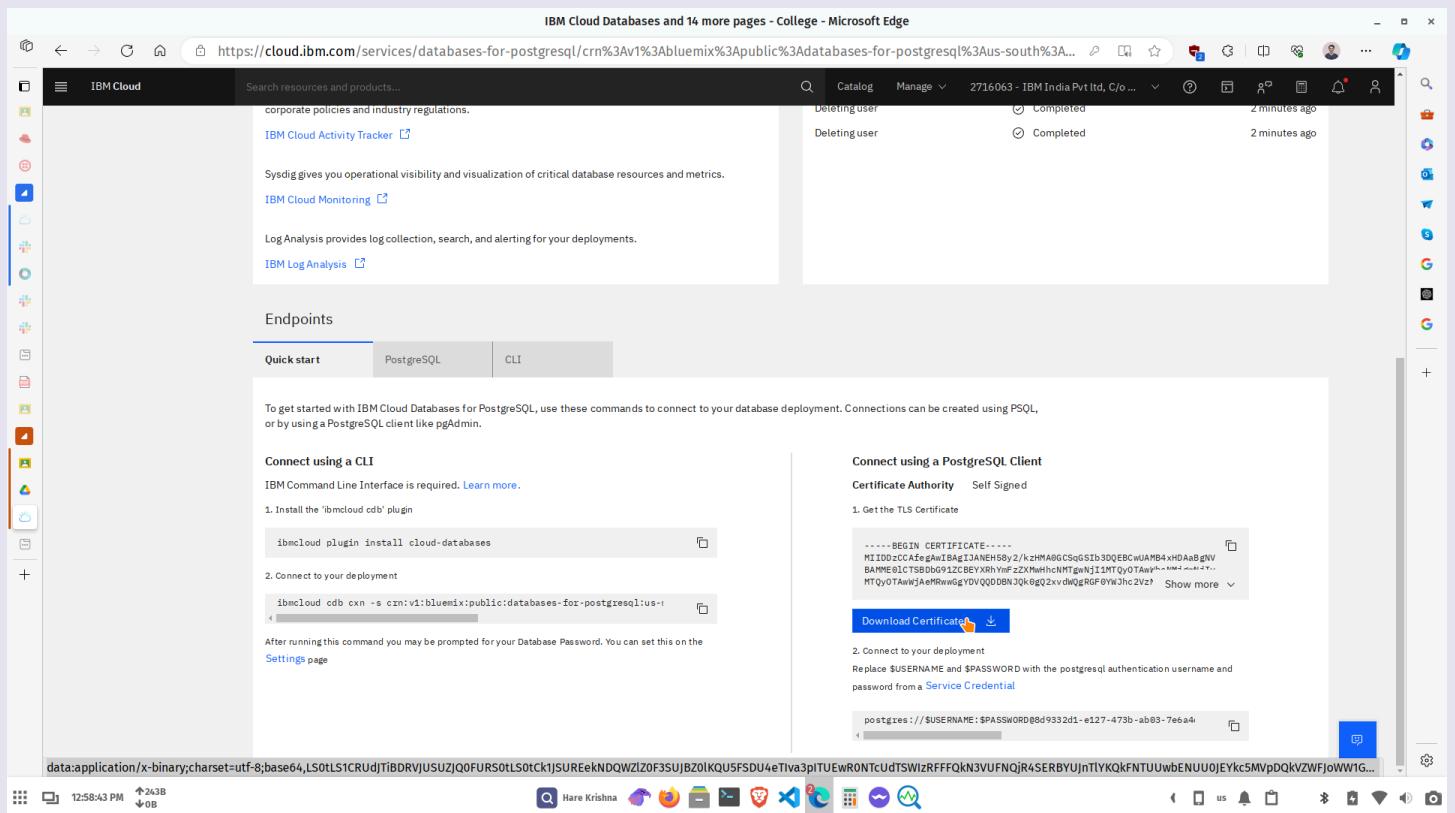
EADC Practical 19

Integration with IBM Cloud :

1. Create if not exists, the PostgreSQL service on IBM Cloud and create if not exist its service credentials

Name - Yash Lakhtariya
Enrollment number - 21162101012
Branch - CBA Batch - 61
EADC Practical 19

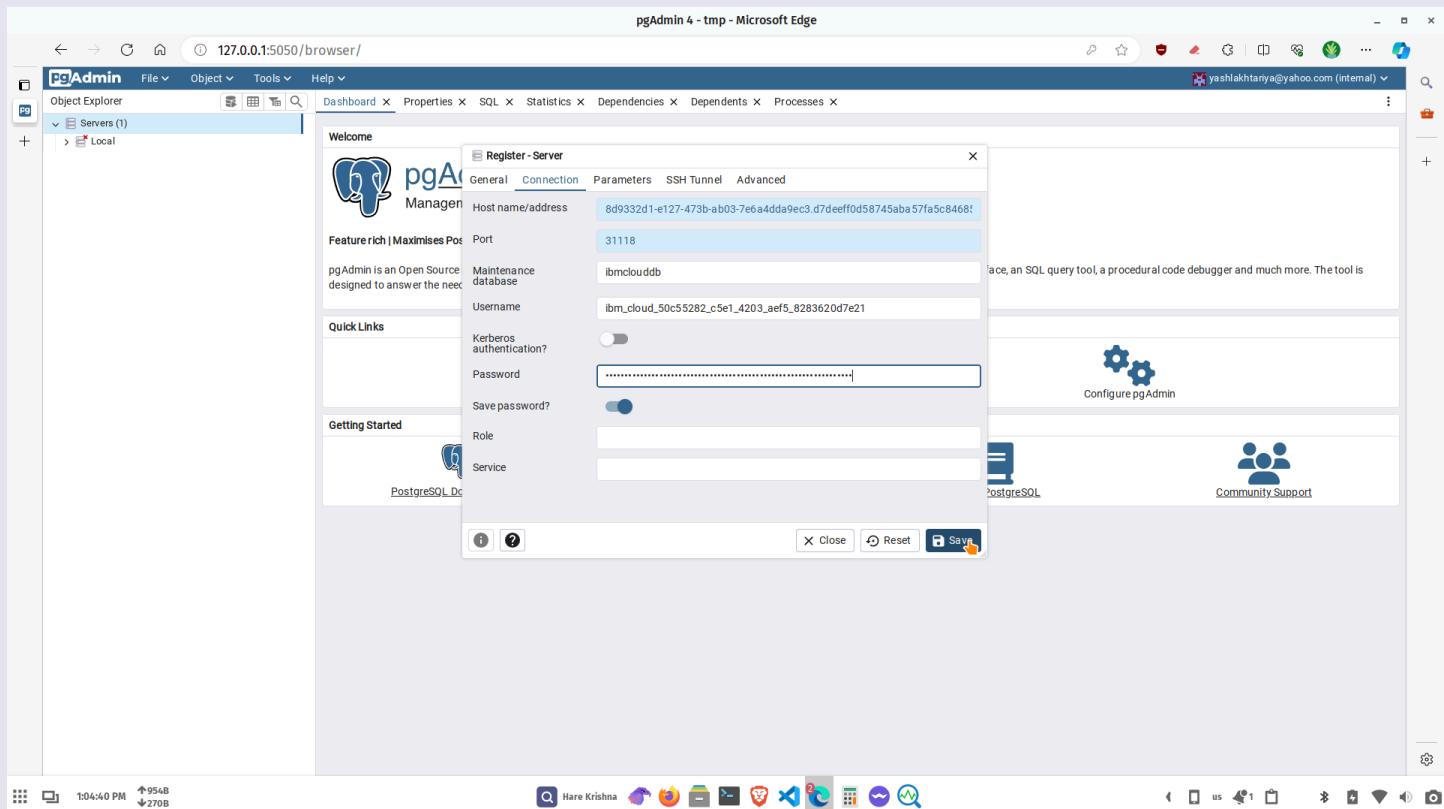
2. Download the TLS certificate to ensure authenticity



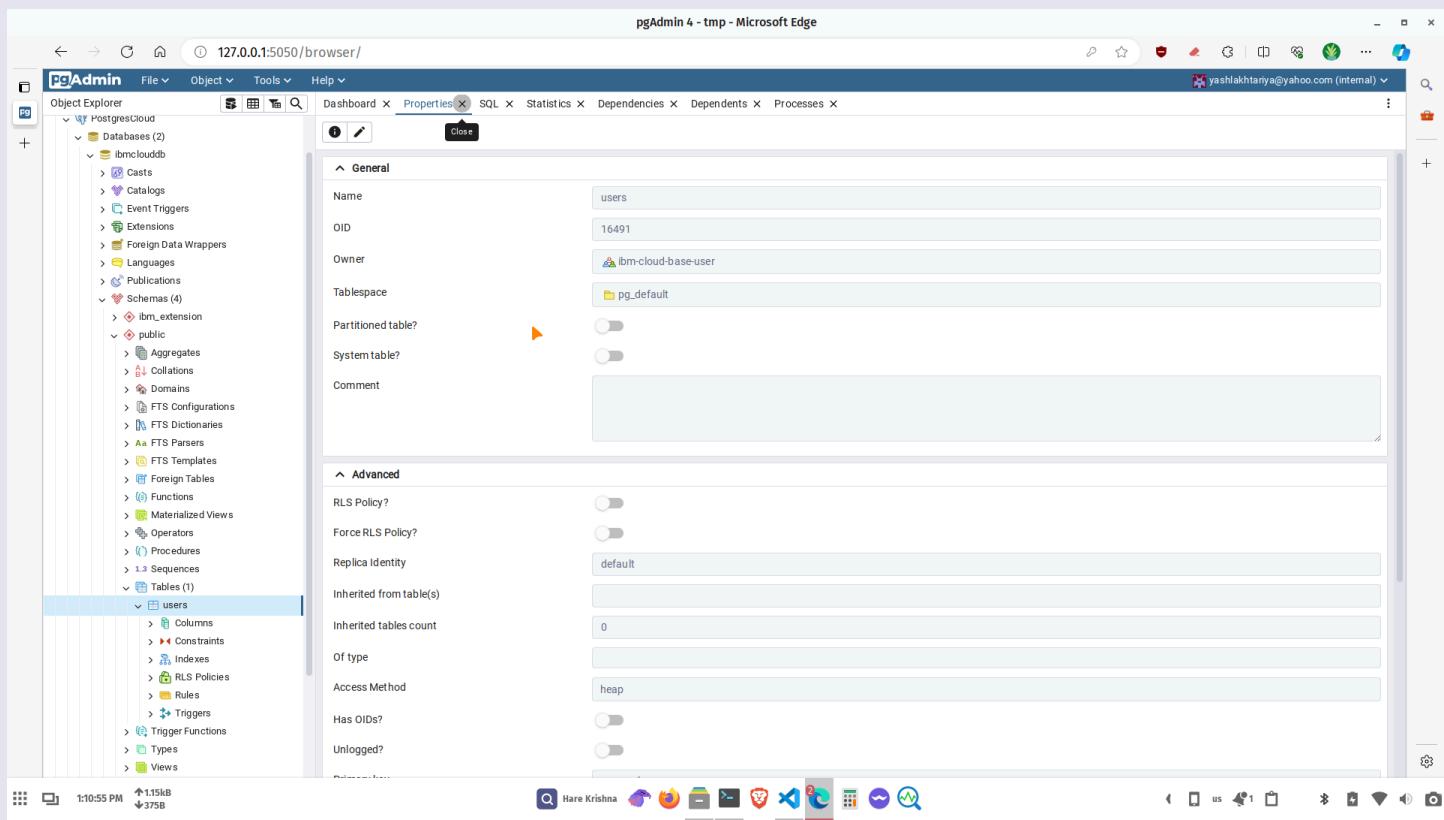
The screenshot shows a Microsoft Edge browser window with the URL <https://cloud.ibm.com/services/databases-for-postgresql/crn%3Av1%3Abluemix%3Apublic%3Adatabases-for-postgresql%3Aus-south%3A...>. The page displays information about the IBM Cloud Databases service, including sections for Corporate Policies and Industry Regulations, IBM Cloud Activity Tracker, Sysdig, IBM Cloud Monitoring, and IBM Log Analysis. Below these, there's a 'Endpoints' section with tabs for 'Quick start', 'PostgreSQL', and 'CLI'. Under 'Quick start', instructions are provided for connecting using a CLI or a PostgreSQL Client. A 'Connect using a PostgreSQL Client' section shows a certificate authority as 'Self Signed' and provides a link to 'Get the TLS Certificate'. A 'Download Certificate' button is highlighted with a mouse cursor. The bottom of the page shows a command-line interface for connecting via the CLI. The browser's address bar, search bar, and various toolbars are visible at the top and bottom of the window.

Name - Yash Lakhtariya
Enrollment number - 21162101012
Branch - CBA Batch - 61
EADC Practical 19

3. Connect the cloud PostgreSQL database with local pgadmin using service credentials : username, password, host, database name and port



Name - Yash Lakhtariya
Enrollment number - 21162101012
Branch - CBA Batch - 61
EADC Practical 19



The screenshot shows the pgAdmin 4 interface with the following details:

- Object Explorer:** Shows the database structure under "Postgrescloud".
 - Databases (2):** ibmclouddb (selected), public
 - Tables (1):** users (selected)
- Properties Tab:** The "users" table is selected. The properties shown are:
 - General:** Name: users, OID: 16491, Owner: ibm-cloud-base-user, Tablespace: pg_default.
 - Advanced:** RLS Policy?, Force RLS Policy?, Replica Identity: default, Inherited from table(s): 0, Inherited tables count: 0, Of type: heap, Access Method: heap, Has OIDs?, Unlogged?
- Bottom Bar:** Shows system icons for battery, signal, volume, and camera.

Name - Yash Lakhtariya

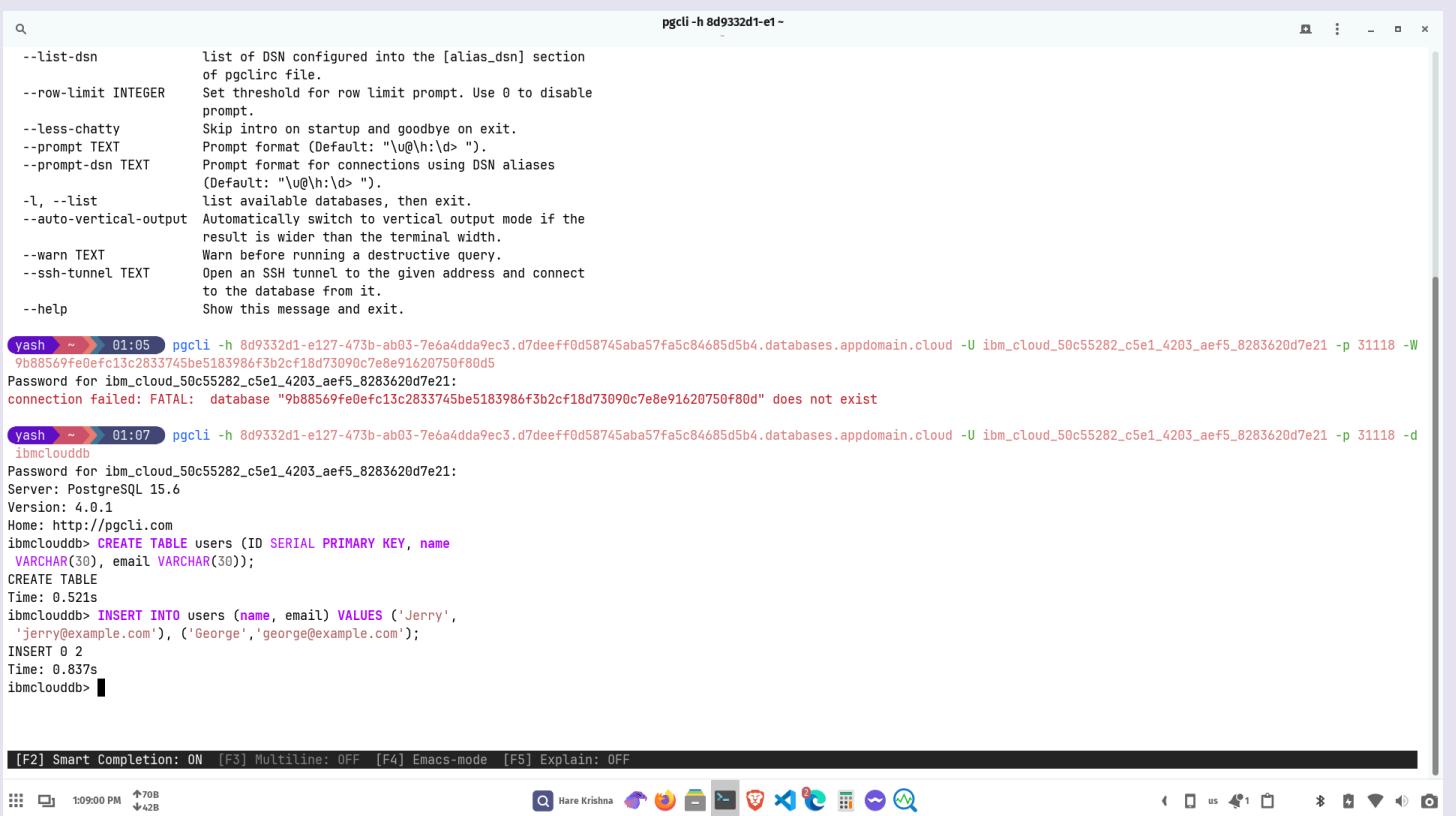
Enrollment number - 21162101012

Branch - CBA Batch - 61

EADC Practical 19

4. Also, it can be connected to local psql command line using the same credentials

(*Here pgcli is used instead of psql, which has same syntax like psql but with features like autocomplete, syntax highlighting, and much more*)



The screenshot shows a terminal window titled "pgcli -h 8d9332d1-e1 ~". It displays the help documentation for pgcli, followed by a connection attempt to a database. The connection fails due to a missing database. Subsequently, a new "ibmclouddb" database is created, and a "users" table is defined and populated with two rows: ('Jerry', 'jerry@example.com') and ('George', 'george@example.com'). The terminal also shows system status at the bottom, including battery level and network activity.

```
pgcli -h 8d9332d1-e1 ~

--list-dsn      list of DSN configured into the [alias_dsn] section
of pgclirc file.
--row-limit INTEGER
Set threshold for row limit prompt. Use 0 to disable
prompt.
--less-chatty
Skip intro on startup and goodbye on exit.
--prompt TEXT
Prompt format (Default: "\u0@\\h:\\d> ").
--prompt-dsn TEXT
Prompt format for connections using DSN aliases
(Default: "\u0@\\h:\\d> ").
-l, --list
list available databases, then exit.
--auto-vertical-output Automatically switch to vertical output mode if the
result is wider than the terminal width.
--warn TEXT
Warn before running a destructive query.
--ssh-tunnel TEXT
Open an SSH tunnel to the given address and connect
to the database from it.
--help
Show this message and exit.

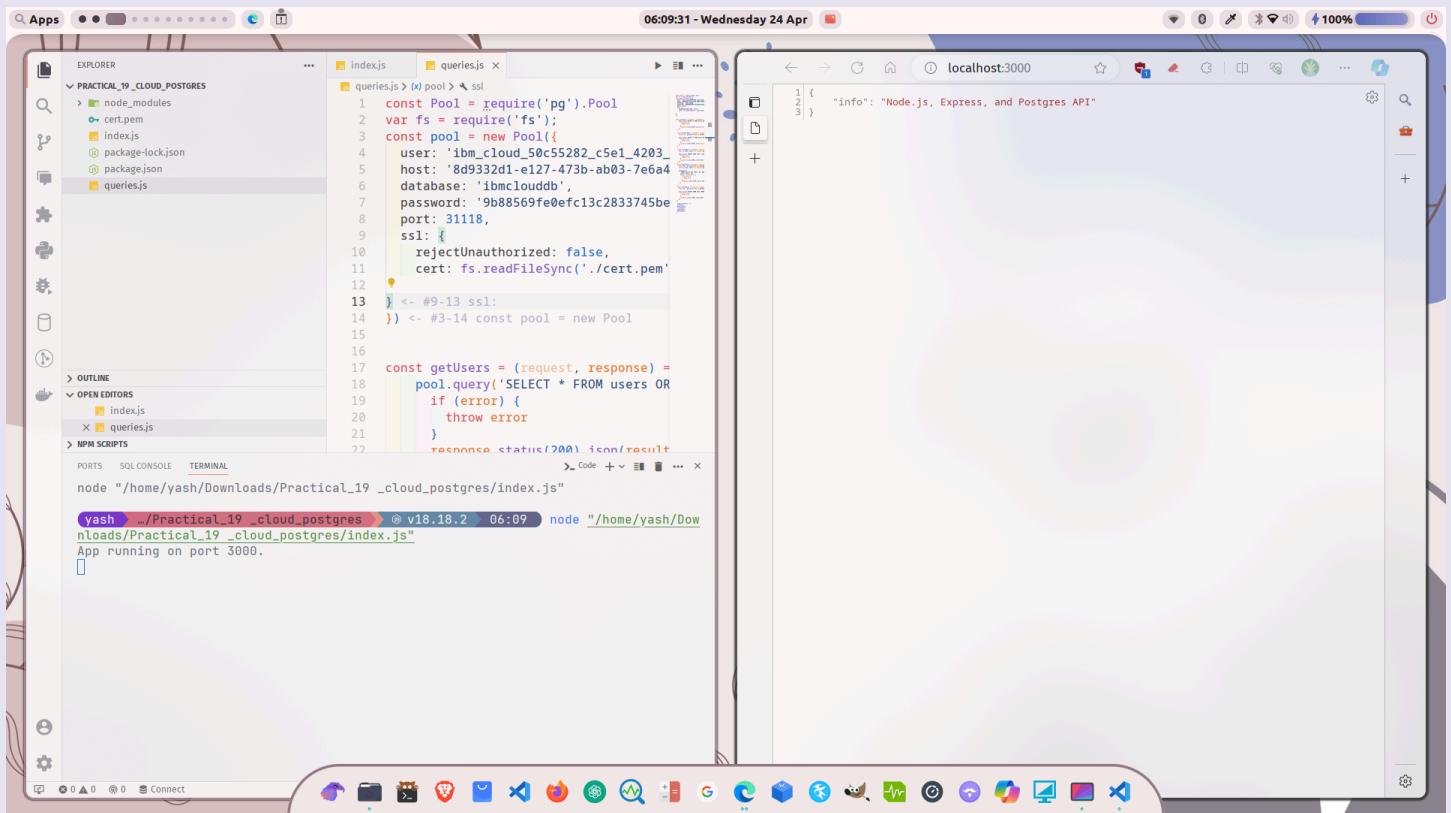
yash ~ 01:05 pgcli -h 8d9332d1-e127-473b-ab03-7e6a4dda9ec3.d7deeff0d58745aba57fa5c84685d5b4.databases.appdomain.cloud -U ibm_cloud_50c55282_c5e1_4203_aef5_8283620d7e21 -p 31118 -W 9b88569fe0efc13c2833745be5183986f3b2cf18d73090c7e8e91620750f80d5
Password for ibm_cloud_50c55282_c5e1_4203_aef5_8283620d7e21:
connection failed: FATAL:  database "9b88569fe0efc13c2833745be5183986f3b2cf18d73090c7e8e91620750f80d" does not exist

yash ~ 01:07 pgcli -h 8d9332d1-e127-473b-ab03-7e6a4dda9ec3.d7deeff0d58745aba57fa5c84685d5b4.databases.appdomain.cloud -U ibm_cloud_50c55282_c5e1_4203_aef5_8283620d7e21 -p 31118 -d ibmclouddb
Password for ibm_cloud_50c55282_c5e1_4203_aef5_8283620d7e21:
Server: PostgreSQL 15.6
Version: 4.0.1
Home: http://pgcli.com
ibmclouddb> CREATE TABLE users (ID SERIAL PRIMARY KEY, name
VARCHAR(30), email VARCHAR(30));
CREATE TABLE
Time: 0.521s
ibmclouddb> INSERT INTO users (name, email) VALUES ('Jerry',
'jerry@example.com'), ('George', 'george@example.com');
INSERT 0 2
Time: 0.837s
ibmclouddb>

[F2] Smart Completion: ON [F3] Multiline: OFF [F4] Emacs-mode [F5] Explain: OFF
```

Name - Yash Lakhtariya
Enrollment number - 21162101012
Branch - CBA Batch - 61
EADC Practical 19

5. Now, integrate it with nodejs application and try running the same Postman requests



Code (queries.js) :

```
const Pool = require('pg').Pool
var fs = require('fs');
const pool = new Pool({
  user: 'ibm_cloud_50c55282_c5e1_4203_aef5_8283620d7e21',
  host: '8d9332d1-e127-473b-ab03-7e6a4dda9ec3.d7deeff0d58745aba57fa5c84685d5b4.firebaseio.cloud',
  database: 'ibmclouddb',
  password: '9b88569fe0efc13c2833745be5183986f3b2cf18d73090c7e8e91620750f80d5',
  port: 31118,
```

Name - Yash Lakhtariya
Enrollment number - 21162101012
Branch - CBA Batch - 61
EADC Practical 19

```
ssl: {
  rejectUnauthorized: false,
  cert: fs.readFileSync('./cert.pem').toString(),
}

const getUsers = (request, response) => {
  pool.query('SELECT * FROM users ORDER BY id ASC', (error, results) => {
    if (error) {
      throw error
    }
    response.status(200).json(results.rows)
  })
}

const getUserById = (request, response) => {
  const id = parseInt(request.params.id)
  pool.query('SELECT * FROM users WHERE id = $1', [id], (error, results) => {
    if (error) {
      throw error
    }
    response.status(200).json(results.rows)
  })
}

const createUser = (request, response) => {
  const { name, email } = request.body
  pool.query('INSERT INTO users (name, email) VALUES ($1, $2) RETURNING *', [name, email], (error, results) => {
    if (error) {
      throw error
    }
    response.status(201).json(results.rows)
  })
}
```

Name - Yash Lakhtariya
Enrollment number - 21162101012
Branch - CBA Batch - 61
EADC Practical 19

```
        throw error
    }
    response.status(201).send(`User added with ID:
${results.rows[0].id}`)
})
}

const updateUser = (request, response) => {
    const id = parseInt(request.params.id)
    const { name, email } = request.body
    pool.query(
        'UPDATE users SET name = $1, email = $2 WHERE id = $3',
        [name, email, id],
        (error, results) => {
            if (error) {
                throw error
            }
            response.status(200).send(`User modified with ID: ${id}`)
        }
    )
}

const deleteUser = (request, response) => {
    const id = parseInt(request.params.id)
    pool.query('DELETE FROM users WHERE id = $1', [id], (error, results)
=> {
        if (error) {
            throw error
        }
        response.status(200).send(`User deleted with ID: ${id}`)
    })
}

module.exports = {
    getUsers,
```

Name - Yash Lakhtariya
Enrollment number - 21162101012
Branch - CBA Batch - 61
EADC Practical 19

```
getUserById,  
createUser,  
updateUser,  
deleteUser,  
}
```

Code (index.js) :

```
const express = require('express')  
const bodyParser = require('body-parser')  
const app = express()  
const db = require('./queries')  
const port = 3000  
  
app.use(bodyParser.json())  
app.use(  
  bodyParser.urlencoded({  
    extended: true,  
  })  
)  
  
app.get('/', (request, response) => {  
  response.json({ info: 'Node.js, Express, and Postgres API' })  
})  
  
app.get('/users', db.getUsers)  
app.get('/users/:id', db.getUserById)  
app.post('/users', db.createUser)  
app.put('/users/:id', db.updateUser)  
app.delete('/users/:id', db.deleteUser)  
  
app.listen(port, () => {  
  console.log(`App running on port ${port}.`)
```

Name - Yash Lakhtariya
Enrollment number - 21162101012
Branch - CBA Batch - 61
EADC Practical 19

)

→ Requests same as above, in Postman, URL, body, same, because changes to integrate with cloud database is made in nodejs code, our API endpoints and output are same :

a) GET all users

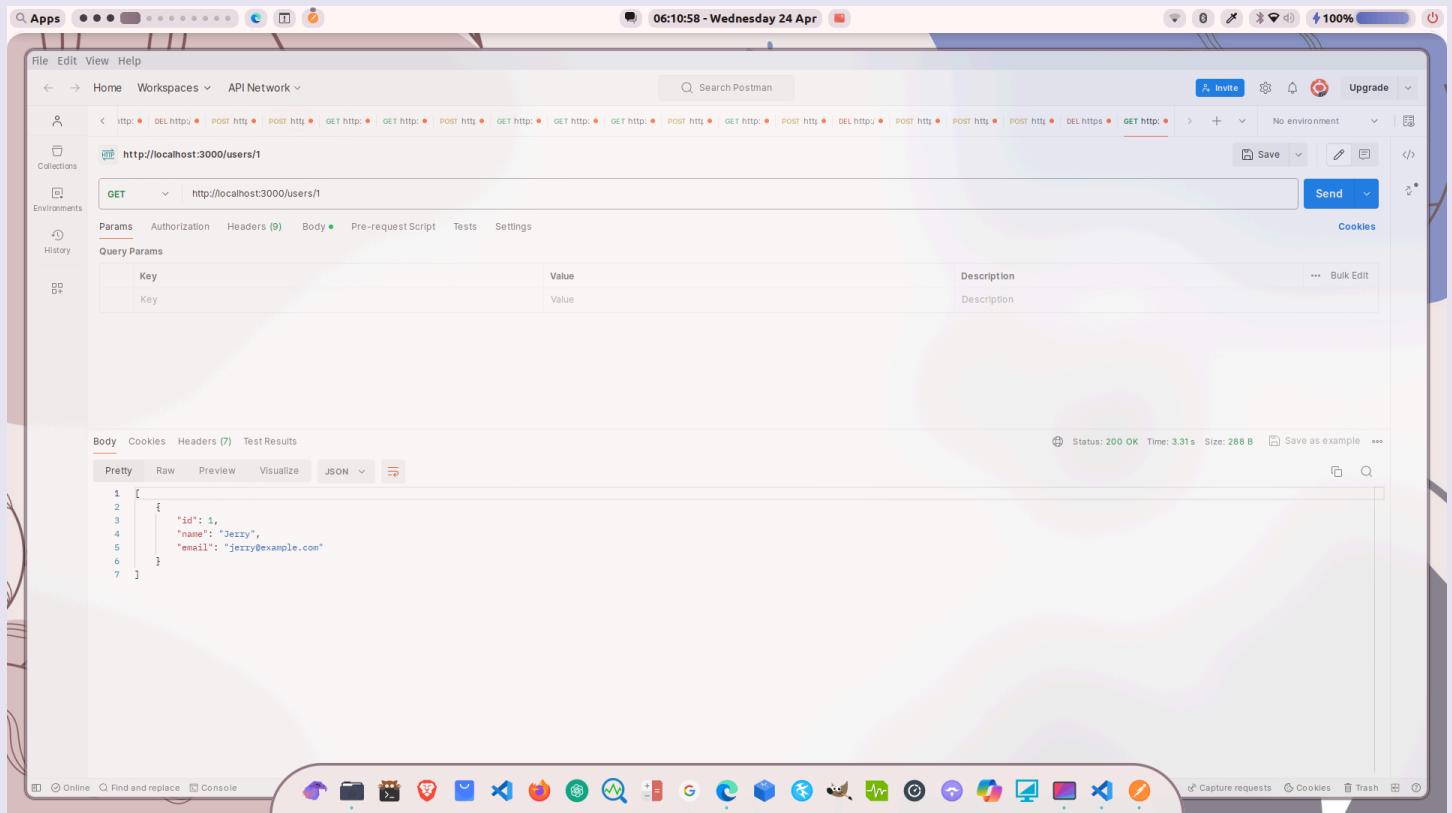
The screenshot shows the Postman application interface. The URL in the address bar is `http://localhost:3000/users/`. The method is set to `GET`. In the 'Body' tab, the response is displayed as JSON:

```
1 [  
2   {  
3     "id": 1,  
4     "name": "Jerry",  
5     "email": "jerry@example.com"  
6   },  
7   {  
8     "id": 3,  
9     "name": null,  
10    "email": null  
11  },  
12  {  
13    "id": 4,  
14    "name": "using_ibmcloud",  
15    "email": "george@example.com"  
16  },  
17  {  
18    "id": 5,  
19    "name": "Hetvi",  
20    "email": "hetvi@example.com"  
21  }]
```

The status bar at the bottom indicates the response was successful (200 OK), took 3.68 seconds, and had a size of 437 B.

Name - Yash Lakhtariya
Enrollment number - 21162101012
Branch - CBA Batch - 61
EADC Practical 19

b) GET details of a user using ID



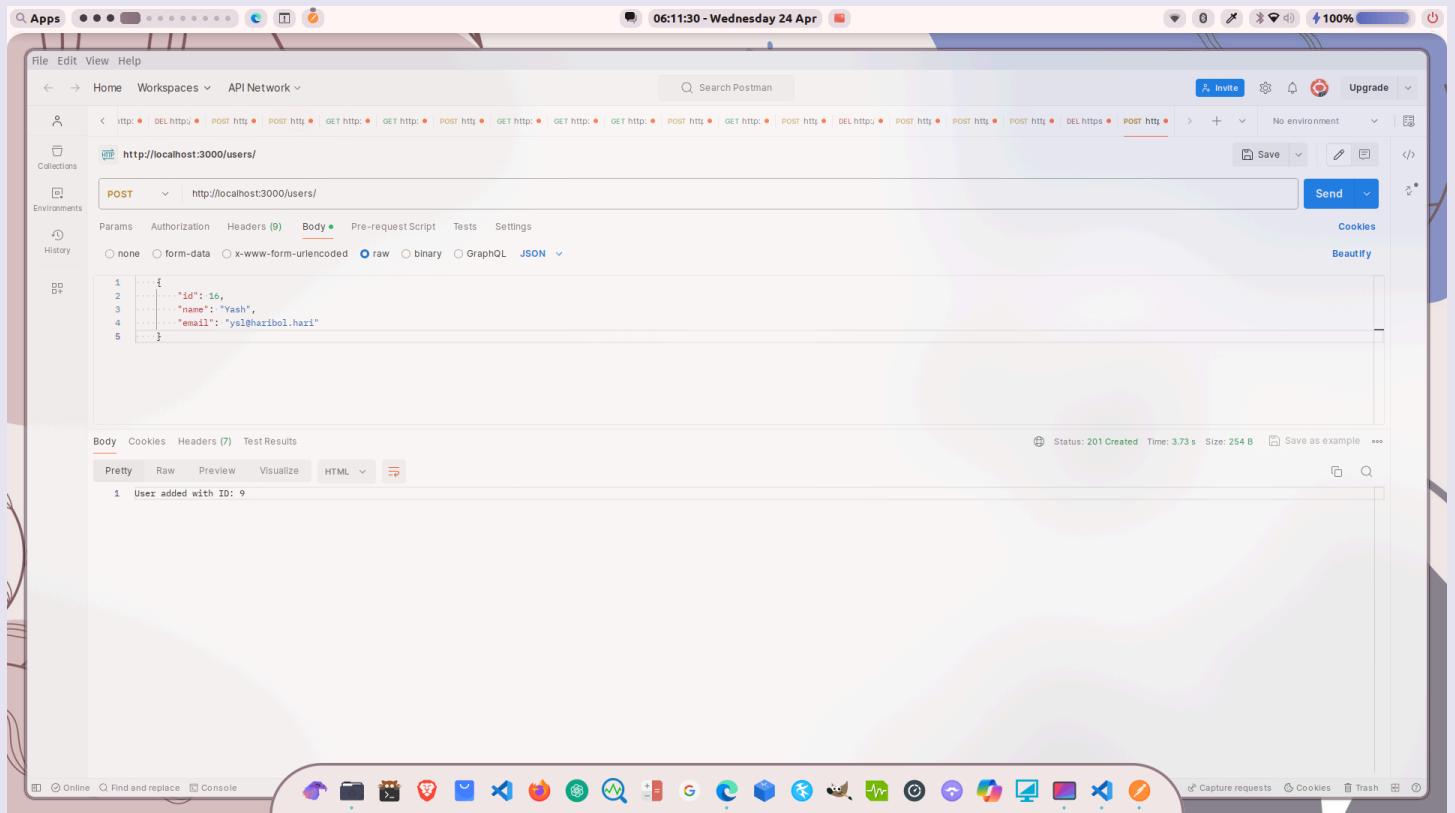
The screenshot shows the Postman application interface. The URL in the header is `http://localhost:3000/users/1`. The request method is set to `GET`. In the 'Query Params' section, there is one entry: 'Key' with a value of 'Value'. The 'Body' tab is selected, showing a JSON response:

```
1 [  
2   {  
3     "id": 1,  
4     "name": "Jerry",  
5     "email": "jerry@example.com"  
6   }  
7 ]
```

The status bar at the bottom indicates a `200 OK` status, a `Time: 3.31s`, and a `Size: 288 B`.

Name - Yash Lakhtariya
Enrollment number - 21162101012
Branch - CBA Batch - 61
EADC Practical 19

c) POST or add a user



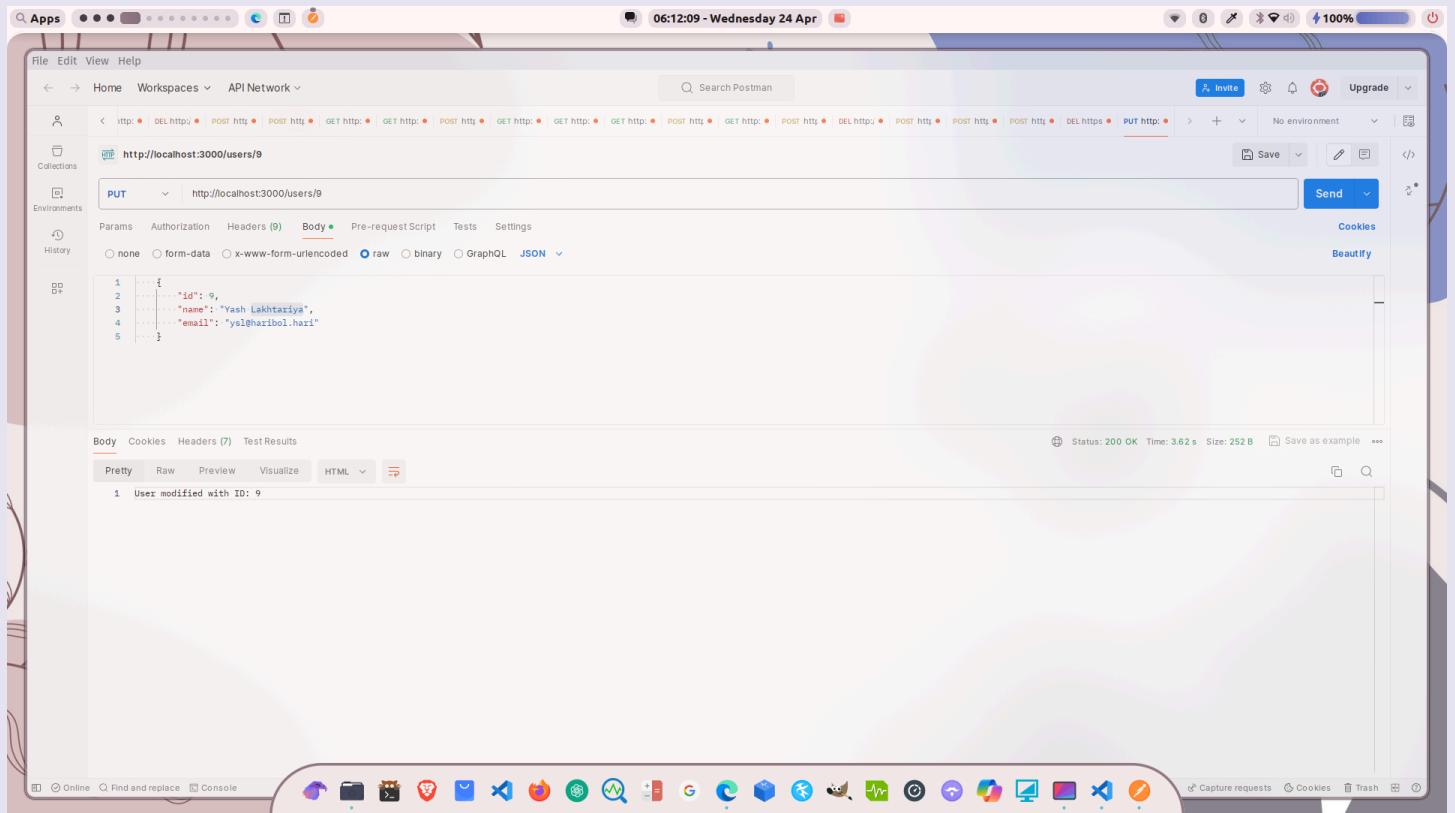
The screenshot shows the Postman application interface. The URL in the header is `http://localhost:3000/users/`. The method selected is `POST`. The request body is set to `JSON` and contains the following JSON data:

```
1 ... {
2 ...     "id": 16,
3 ...     "name": "Yash",
4 ...     "email": "yash@haribol.hari"
5 ... }
```

The response status is `201 Created`, time taken is `3.73 s`, and size is `254 B`. The response body is: `User added with ID: 9`.

Name - Yash Lakhtariya
Enrollment number - 21162101012
Branch - CBA Batch - 61
EADC Practical 19

d) PUT or update a user



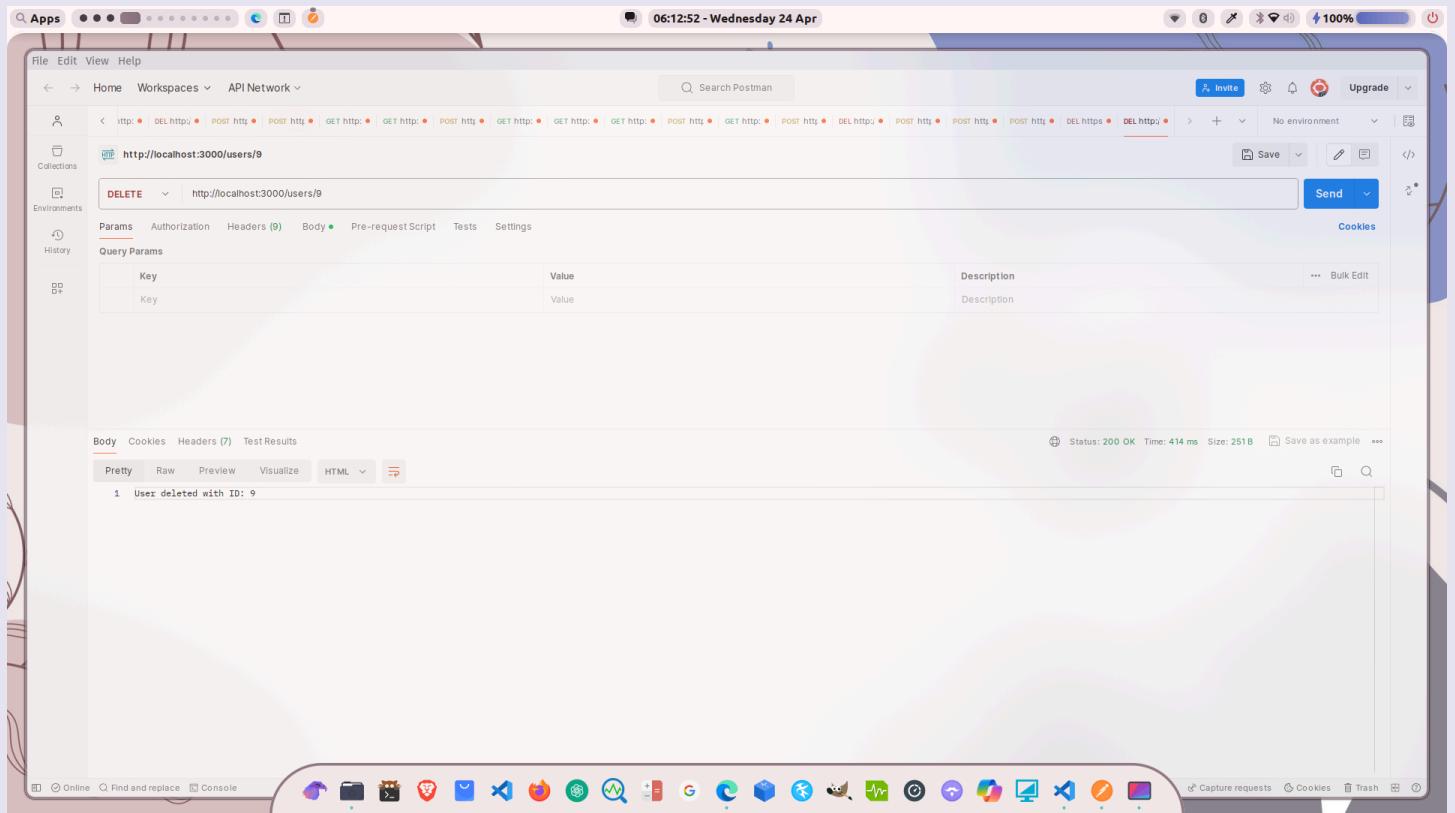
The screenshot shows the Postman application interface. The URL in the header is `http://localhost:3000/users/9`. The method selected is `PUT`. The request body is set to `JSON` and contains the following JSON data:

```
1 ... {
2 ...     "id": 9,
3 ...     "name": "Yash Lakhtariya",
4 ...     "email": "ysl@haribol.hari"
5 ... }
```

In the response section, the status is `200 OK`, time is `3.62 s`, and size is `252 B`. The message `User modified with ID: 9` is displayed.

Name - Yash Lakhtariya
Enrollment number - 21162101012
Branch - CBA Batch - 61
EADC Practical 19

e) DELETE a user



The screenshot shows the Postman application interface. The URL in the address bar is `http://localhost:3000/users/9`. The method selected is `DELETE`. In the "Query Params" section, there is one entry: "Key" with "Value". The "Body" tab is selected, showing the response: "User deleted with ID: 9". The status bar at the bottom indicates "Status: 200 OK Time: 414 ms Size: 251 B".