University of East Anglia
COMPUTING SCIENCES

Lab 9: Postgres, Node, and Forms

Before attempting this lab, you will need to have completed **Lab 9: Forms and Web Security**

Learning objectives

By completing this lab, you should be able to:

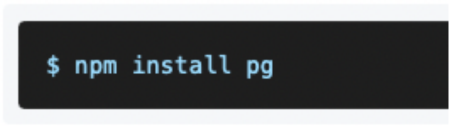
- Install Node Postgres package on a local machine
- Run SQL queries using pgAdmin4 interface
- Write server code to connect to Postgres server
- Write server code to insert data to Postgres database

Background

We will be using `node-postgres` package for this exercise.

`node-postgres` is a collection of Node modules for interfacing with PostgreSQL database. It has support for callbacks, promises, async/await, connection pooling, prepared statements, cursors, streaming results and other PostgreSQL features.

If you have not installed this library on your machine, then you should install it now:



```
$ npm install pg
```

Figure 1. Install node-postgres

Database design and implementation is not covered on this module. You should be familiar with using PostgreSQL. Both are taught in Database Manipulation module (CMP-7025A) so I am not going to repeat these topics but instead we are going to jump straight in to the tasks.

Exercises

1. Create a new directory in your Document directory called `postgres-node`. Copy all the files in `node-express` directory to this new directory.

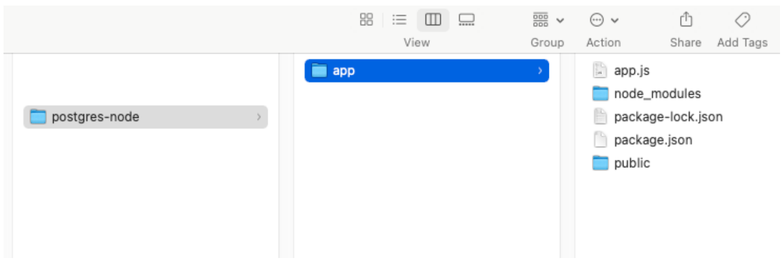


Figure 2. This week lab folder

2. Using pgAdmin4 and run SQL queries
   a) Type `postgresql` in the Windows search bar and click on `Postgresql` to start PostgreSQL manager.

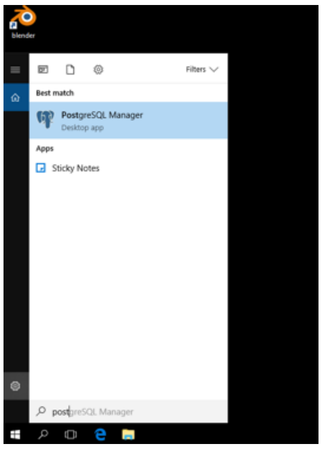
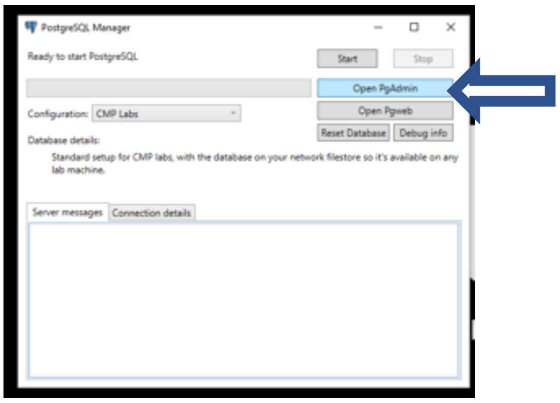


Figure 3. Windows search bar

   b) Click the "OpenPgAdmin" button on the PostgreSQL manager UI as shown below:



Ready to start PostgreSQL

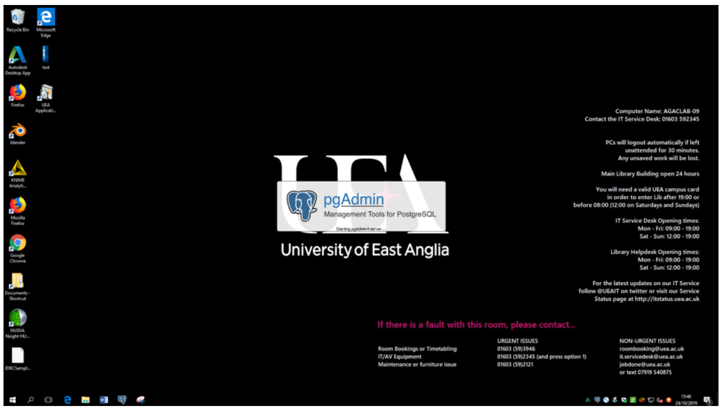
Configuration: CMP Labs

Database details:

Standard setup for CMP labs, with the database on your network filestore so it's available on any lab machine.

Figure 4. Windows dialog box

It may take a minute or two to load the pgAdmin4, just be patient.


pgAdmin
Management Tools for PostgreSQL
Starting pgAdmin4 server...
University of East Anglia
15:46
24/10/2019

Computer Name: AGACLAB-09
Contact the IT Service Desk: 01603 592345

PCs will logout automatically if left
unattended for 30 minutes.
Any unsaved work will be lost.

Main Library Building open 24 hours

You will need a valid UEA campus card
in order to enter Lib after 19:00 or
before 08:00 (12:00 on Saturdays and Sundays)

IT Service Desk Opening times:
Mon - Fri: 09:00 - 19:00
Sat - Sun: 12:00 - 19:00

Library Helpdesk Opening times:
Mon - Fri: 09:00 - 19:00
Sat - Sun: 12:00 - 19:00

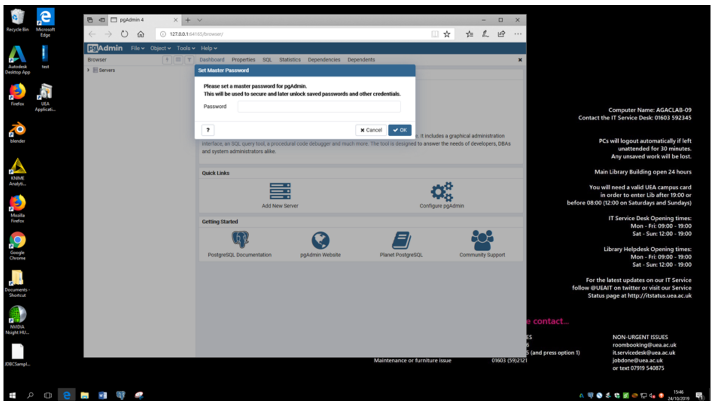
For the latest updates on our IT Service
follow @UEAIT on twitter or visit our Service
Status page at http://itstatus.uea.ac.uk

If there is a fault with this room, please contact...

	URGENT ISSUES	NON-URGENT ISSUES
Room Bookings or Timetabling	01603 (59)3946	roombooking@uea.ac.uk
IT/AV Equipment	01603 (59)2345 (and press option 1)	it.servicedesk@uea.ac.uk
Maintenance or furniture issue	01603 (59)2121	jobdone@uea.ac.uk
		or text 07919 540875

Figure 5. Windows desktop view

c) Once the pgAdmin4 has loaded, you will be prompted to enter a password, type "12345" in the password box

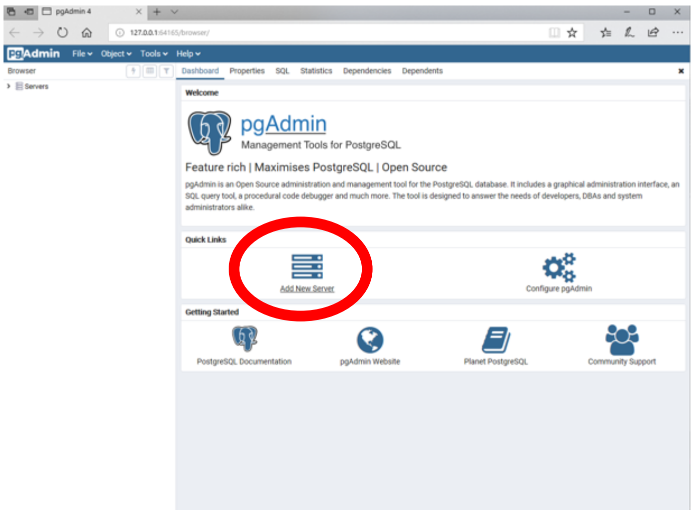

Set Master Password

Please set a master password for pgAdmin.
This will be used to secure and later unlock saved passwords and other credentials.

Password

Figure 6. Windows desktop view

d) Once this is done, the pgAdmin4 UI will appear on the screen. Click the "Add New Server" icon in the "Quick Links" section


Welcome

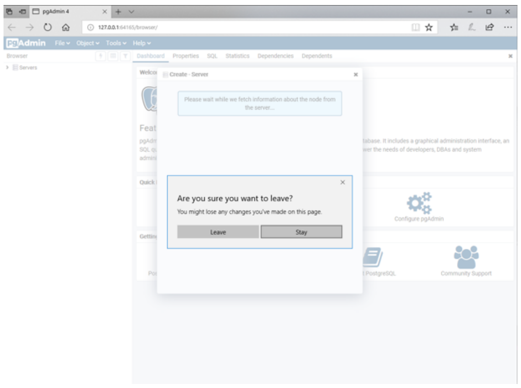
pgAdmin
Management Tools for PostgreSQL

Feature rich | Maximises PostgreSQL | Open Source

pgAdmin is an Open Source administration and management tool for the PostgreSQL database. It includes a graphical administration interface, an SQL query tool, a procedural code debugger and much more. The tool is designed to answer the needs of developers, DBAs and system administrators alike.

Figure 7. Windows desktop view

e) A screen might appear to ask whether you wish to leave the page, just click "Stay".


Create - Server

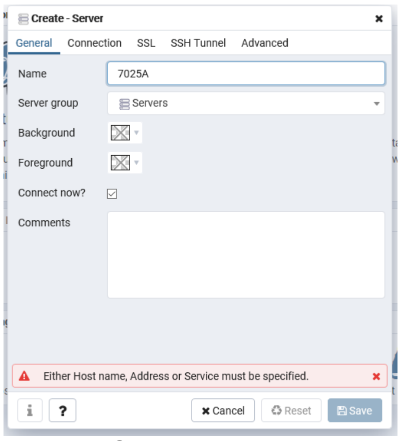
Please wait while we fetch information about the node from the server...

Are you sure you want to leave?

You might lose any changes you've made on this page.

Figure 8. Windows desktop view

f) Next, we will configure our new server information. Enter "7003A" as the name of this server.


Create - Server

General | Connection | SSL | SSH Tunnel | Advanced

Name: 7025A
Server group: Servers
Background:
Foreground:
Connect now? ✓
Comments:

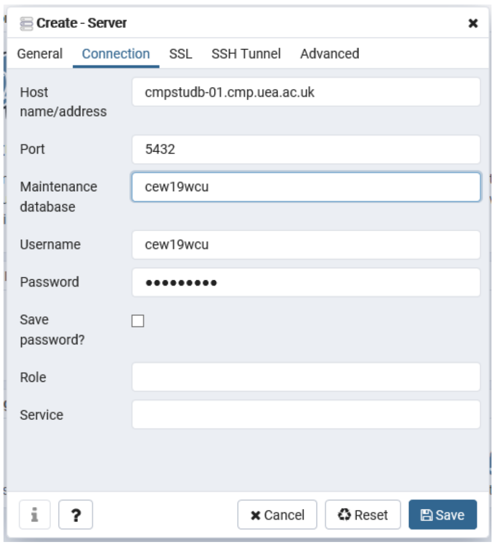
Either Host name, Address or Service must be specified.

Figure 9. Create a new Postgres server

g) Next, click the "Connection" tab, and enter the following server information in appropriate boxes:

Server hostname: cmpstudb-01.cmp.uea.ac.uk
Database Name: UEA Username
Username: UEA Username
Password: UEA Password

h) The image below (Figure 10) shows my pgAdmin4 server configuration using my UEA login credential. You will need to replace my credential with yours.


Create - Server

General | Connection | SSL | SSH Tunnel | Advanced

Host name/address: cmpstudb-01.cmp.uea.ac.uk
Port: 5432
Maintenance database: cew19wcu
Username: cew19wcu
Password: •••••••••
Save password? ☐
Role:
Service:

Figure 10. Postgres server credentials

i) Click the “Save” button once this is done. Notice that the new server will automatically be appended on the left panel (in the server tree).

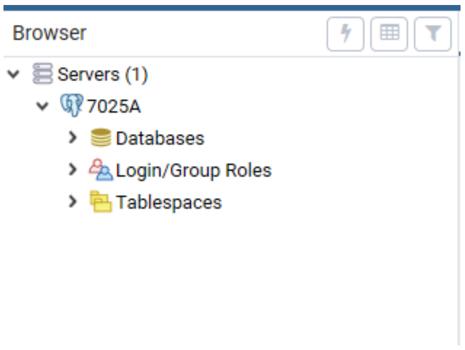


Figure 11. The tree view of the server.

j) Next, expand the “Databases” tree and identify your database (i.e. your username). Below diagram shows my database (i.e. cew19wcu) in the “Databases” tree.

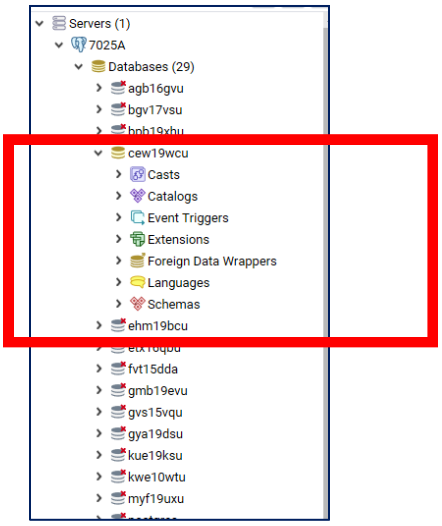


Figure 12. The tree view of the server

k) We will bring up the “Query tool” to write our SQL statements. Right click on the database with your username e.g. “abc20abc” and select “Query Tool”

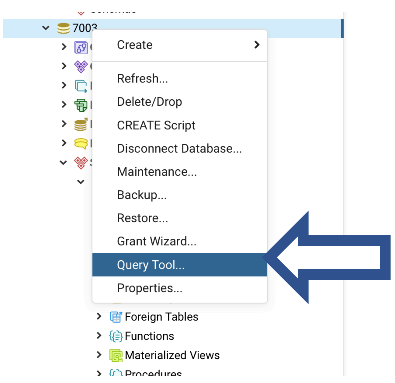


Figure 13. Select Query Tool from the database view

l) A query panel appears on the right side of the window once the “Query Tool” is selected.

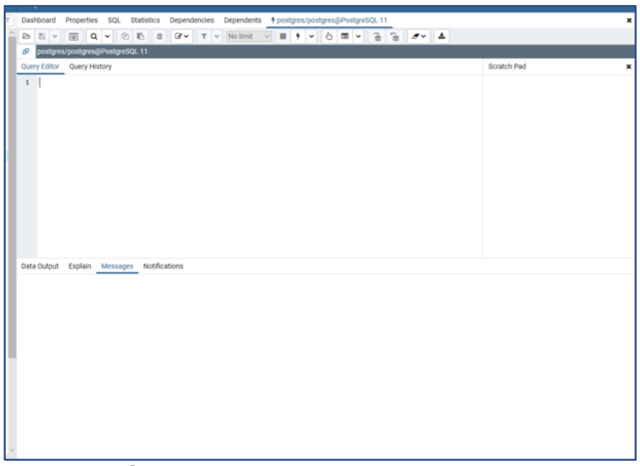
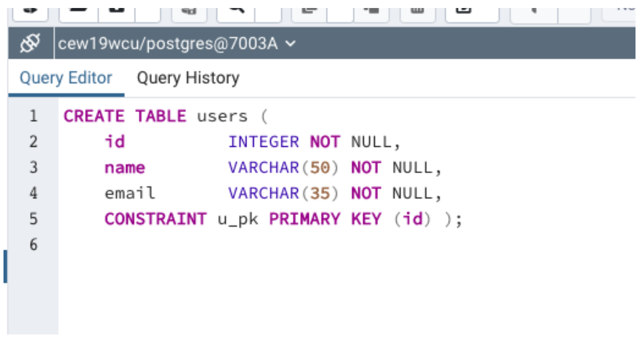


Figure 14. Query panel

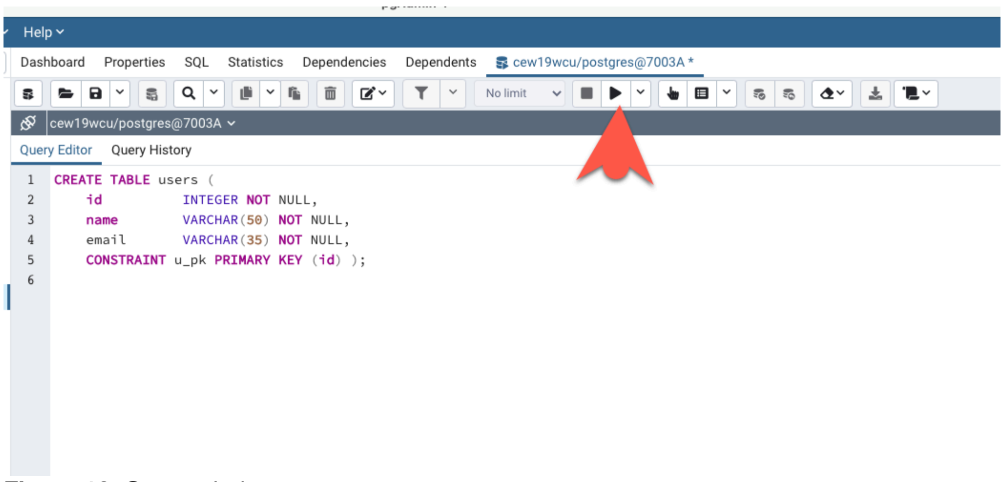
m) In the Query Editor window, enter the following SQL statements:



```
CREATE TABLE users (
    id          INTEGER NOT NULL,
    name        VARCHAR(50) NOT NULL,
    email       VARCHAR(35) NOT NULL,
    CONSTRAINT u_pk PRIMARY KEY (id) );
```

Figure 15. SQL for creating a new table

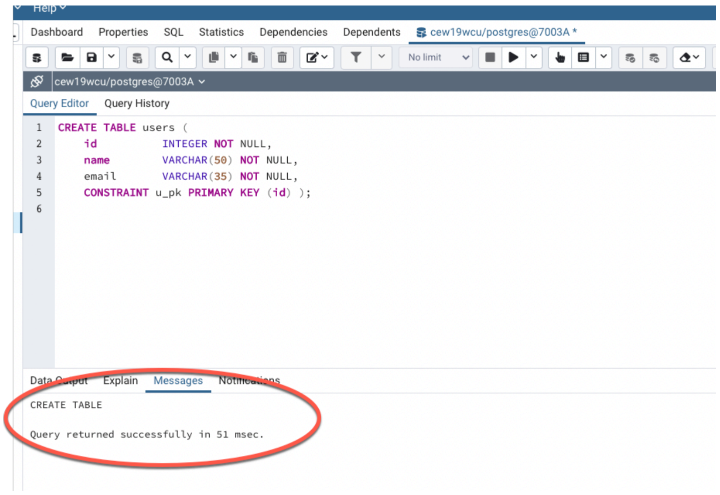
n) And click the ‘right arrow’ icon to run the query:



```
CREATE TABLE users (
    id          INTEGER NOT NULL,
    name        VARCHAR(50) NOT NULL,
    email       VARCHAR(35) NOT NULL,
    CONSTRAINT u_pk PRIMARY KEY (id) );
```

Figure 16. Query window

o) After this you should see Similar results show at the bottom panel:



```
CREATE TABLE users (
    id          INTEGER NOT NULL,
    name        VARCHAR(50) NOT NULL,
    email       VARCHAR(35) NOT NULL,
    CONSTRAINT u_pk PRIMARY KEY (id) );
```

```
CREATE TABLE

Query returned successfully in 51 msec.
```

Figure 17. Query window

p) Now, check to see if the table “users” exists:

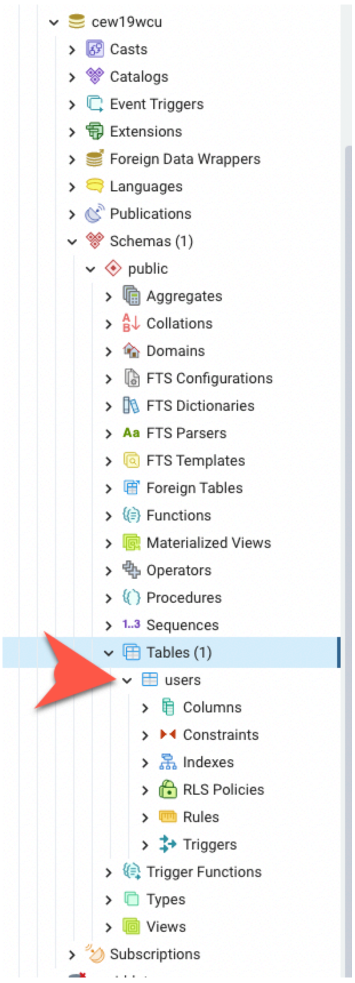
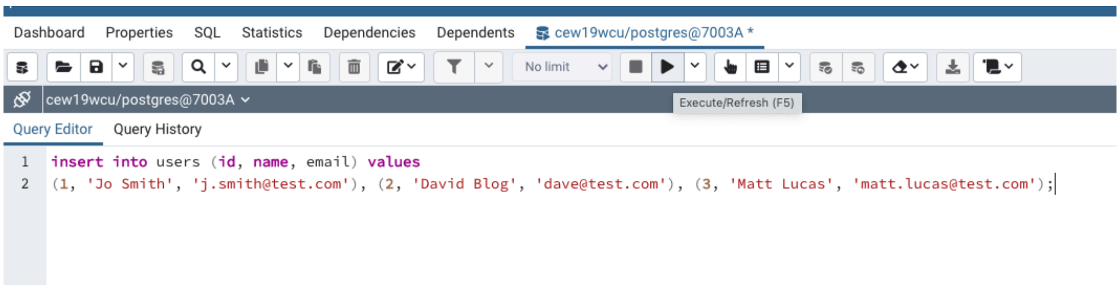


Figure 18. The tree view of the database

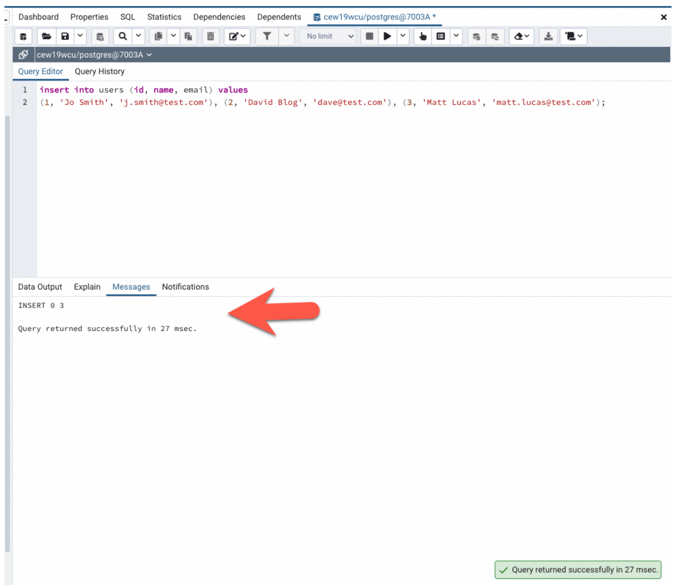
q) Next, we will populate some data. Type in the SQL statements below in the query window and click the “execute” icon (i.e. right arrow icon)



```
insert into users (id, name, email) values
(1, 'Jo Smith', 'j.smith@test.com'), (2, 'David Blog', 'dave@test.com'), (3, 'Matt Lucas', 'matt.lucas@test.com');
```

Figure 19. SQL statements

r) Check the results:



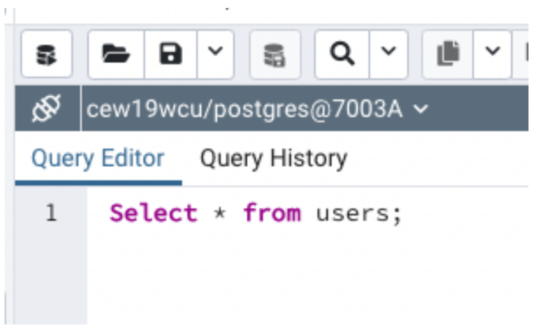
```
insert into users (id, name, email) values
(1, 'Jo Smith', 'j.smith@test.com'), (2, 'David Blog', 'dave@test.com'), (3, 'Matt Lucas', 'matt.lucas@test.com');
```

```
INSERT 0 3

Query returned successfully in 27 msec.
```

✓ Query returned successfully in 27 msec.

Figure 20. Check the results

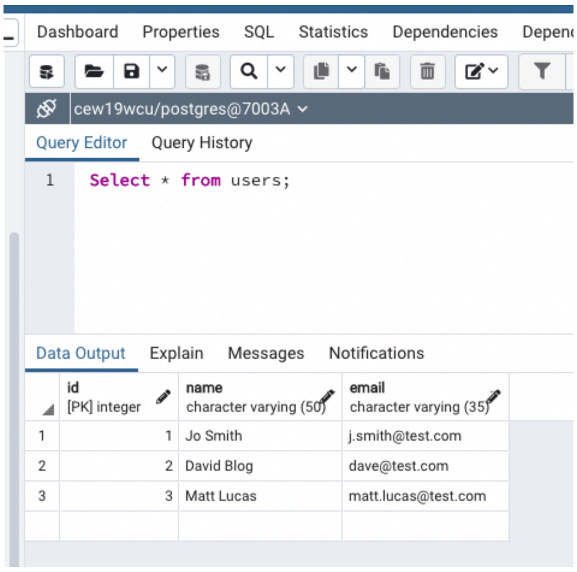
s) Now try the following query in the query window:



```
Select * from users;
```

Figure 21. Execute SQL query

t) Check the results:



```
Select * from users;
```

	id [PK] integer	name character varying (50)	email character varying (35)
1	1	Jo Smith	j.smith@test.com
2	2	David Blog	dave@test.com
3	3	Matt Lucas	matt.lucas@test.com

Figure 22. SQL query results

Make sure that you have completed the above database set up and testing. We will write some code to interact with this database next.

3. Write Node functions to connect to PostgresSQL:
   i. On your text editor, create a new file called `config.js` and save this file in the "`postgres-node/app`" directory.

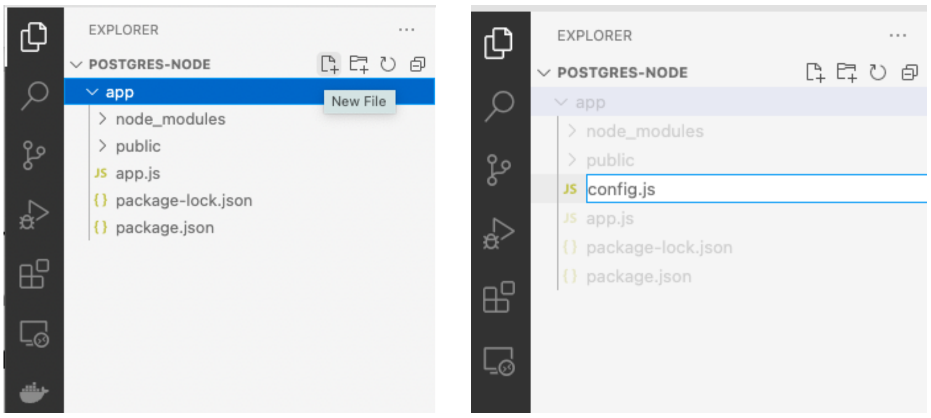
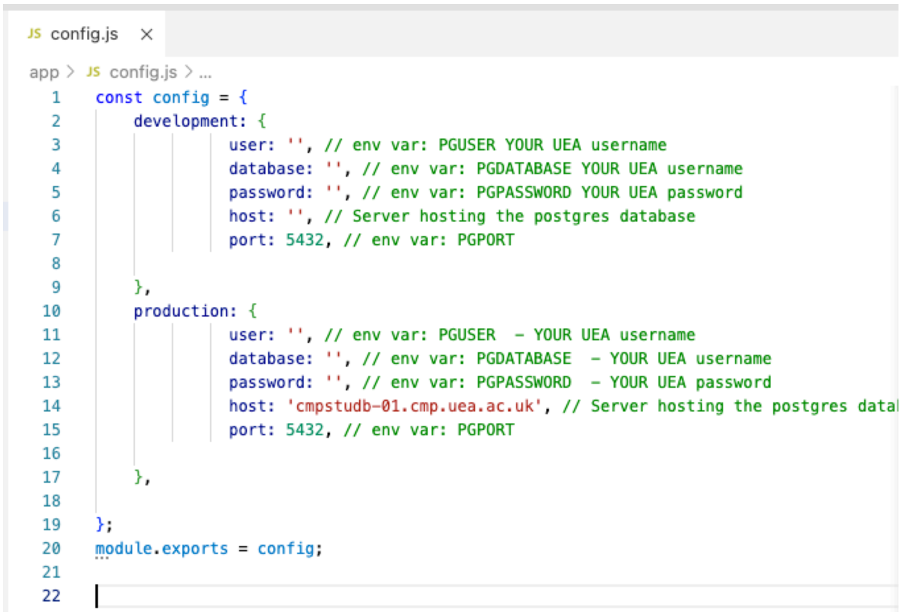


Figure 23. Create a new config.js file

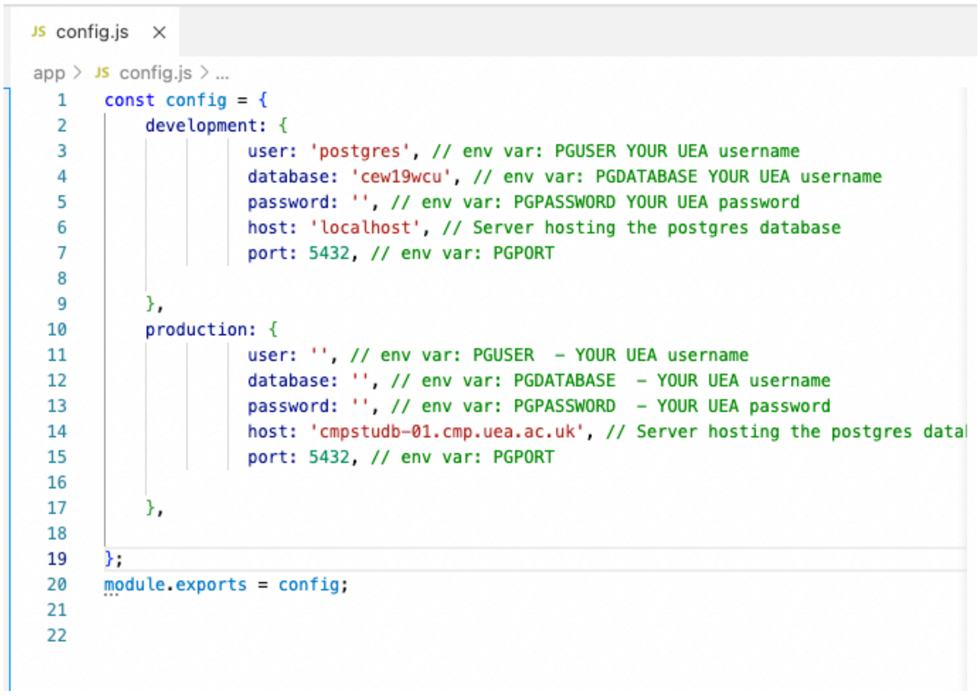
ii. Type below configurations in your `config.js` file



```
config.js ×
app > config.js > ...
const config = {
    development: {
            user: '', // env var: PGUSER YOUR UEA username
            database: '', // env var: PGDATABASE YOUR UEA username
            password: '', // env var: PGPASSWORD YOUR UEA password
            host: '', // Server hosting the postgres database
            port: 5432, // env var: PGPORT

    },
    production: {
            user: '', // env var: PGUSER  - YOUR UEA username
            database: '', // env var: PGDATABASE  - YOUR UEA username
            password: '', // env var: PGPASSWORD  - YOUR UEA password
            host: 'cmpstudb-01.cmp.uea.ac.uk', // Server hosting the postgres data
            port: 5432, // env var: PGPORT

    },

};
module.exports = config;

```

Figure 24. Database configurations

iii. Edit the configurations for "development" on `config.js` **with your details**. **NOTE**: If PostgresSQL is run on your own computer, the host is localhost. Figure 25 below is my details as I am running a local database server.

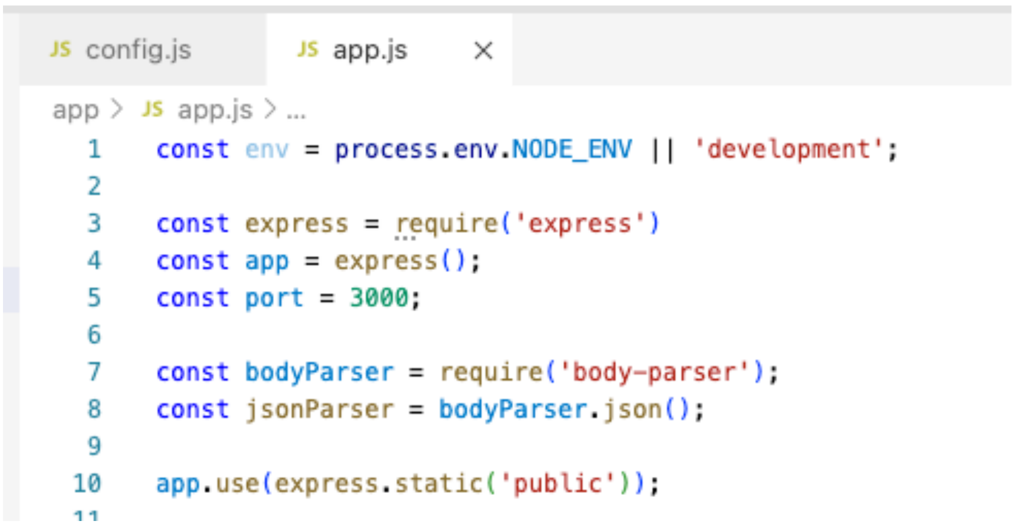


```
config.js ×
app > config.js > ...
const config = {
    development: {
            user: 'postgres', // env var: PGUSER YOUR UEA username
            database: 'cew19wcu', // env var: PGDATABASE YOUR UEA username
            password: '', // env var: PGPASSWORD YOUR UEA password
            host: 'localhost', // Server hosting the postgres database
            port: 5432, // env var: PGPORT

    },
    production: {
            user: '', // env var: PGUSER  - YOUR UEA username
            database: '', // env var: PGDATABASE  - YOUR UEA username
            password: '', // env var: PGPASSWORD  - YOUR UEA password
            host: 'cmpstudb-01.cmp.uea.ac.uk', // Server hosting the postgres data
            port: 5432, // env var: PGPORT

    },

};
module.exports = config;

```

Figure 25. Jeannette Chin local server details

iv. Next, on your text editor, open the `app.js` file, and follow instructions below:

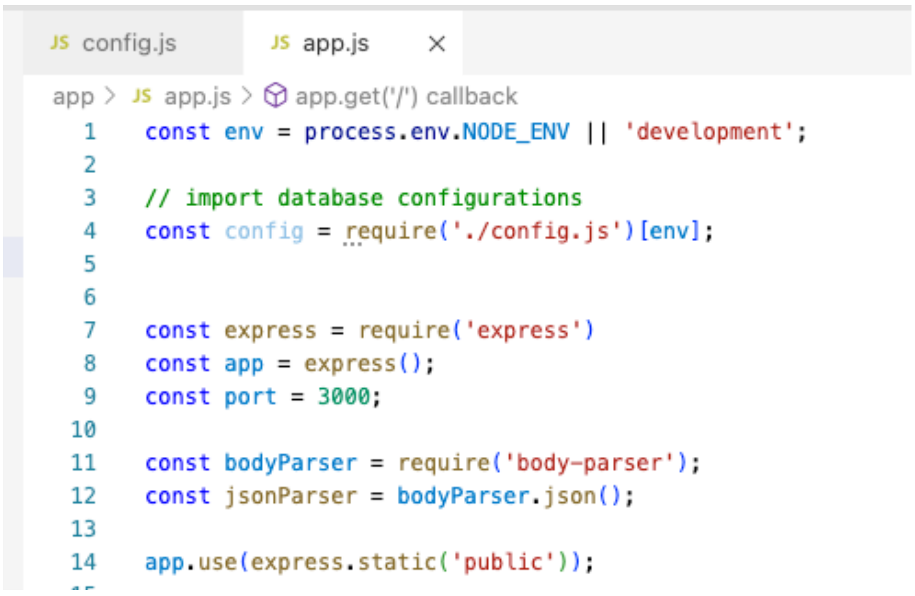
1) First, we will configure our Node to run in development mode – Figure 26, line 1



```
1   const env = process.env.NODE_ENV || 'development';
2
3   const express = require('express')
4   const app = express();
5   const port = 3000;
6
7   const bodyParser = require('body-parser');
8   const jsonParser = bodyParser.json();
9
10  app.use(express.static('public'));
```

Figure 26. The code in app.js

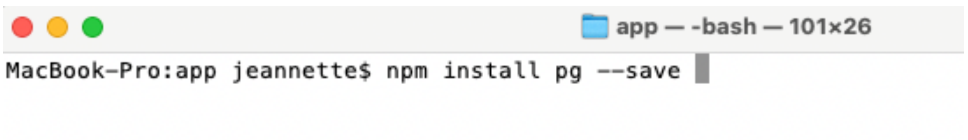
2) Next, we will import the database configurations we have written in `config.js` file – Figure 27, line 4



```
1   const env = process.env.NODE_ENV || 'development';
2
3   // import database configurations
4   const config = require('./config.js')[env];
5
6
7   const express = require('express')
8   const app = express();
9   const port = 3000;
10
11  const bodyParser = require('body-parser');
12  const jsonParser = bodyParser.json();
13
14  app.use(express.static('public'));
```

Figure 27. The code in app.js

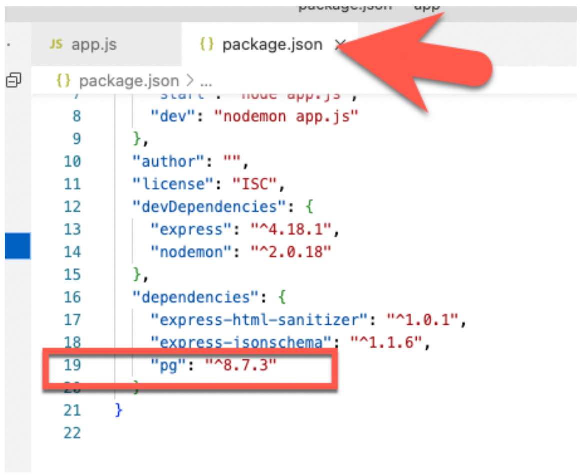
3) Make sure that you have `node-postgres` dependency installed. If not run this command on a terminal window in your project root directory:



```
MacBook-Pro:app jeannette$ npm install pg --save
```

Figure 28. run a command to install pg package

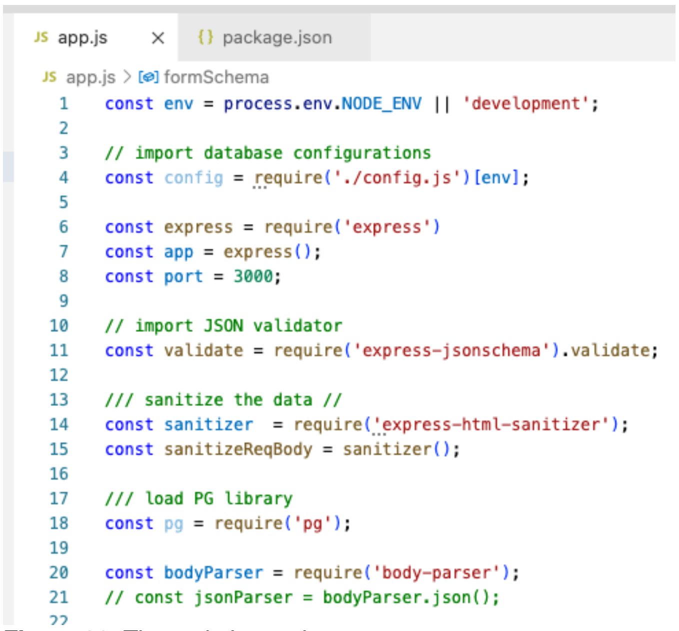
After this you should see the dependencies in your `package.json` file:



```
    "dev": "nodemon app.js"
  },
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "express": "^4.18.1",
    "nodemon": "^2.0.18"
  },
  "dependencies": {
    "express-html-sanitizer": "^1.0.1",
    "express-jsonschema": "^1.1.6",
    "pg": "^8.7.3"
  }
}
```

Figure 29. The content in package.json file

4) Now load the `node-postgres` library in `app.js` – Figure 30, line 18

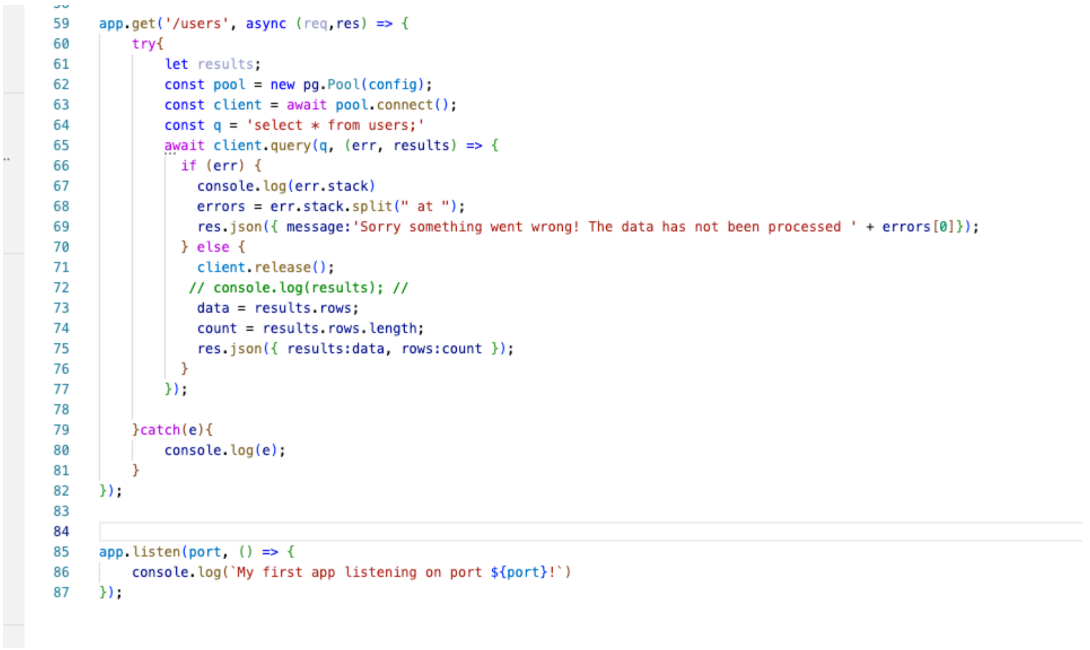


app.js × {} package.json
app.js > formSchema

```javascript
const env = process.env.NODE_ENV || 'development';

// import database configurations
const config = require('./config.js')[env];

const express = require('express')
const app = express();
const port = 3000;

// import JSON validator
const validate = require('express-jsonschema').validate;

/// sanitize the data //
const sanitizer  = require('express-html-sanitizer');
const sanitizeReqBody = sanitizer();

/// load PG library
const pg = require('pg');

const bodyParser = require('body-parser');
// const jsonParser = bodyParser.json();
```

Figure 30. The code in app.js

5) Create a new GET route with a path “users” for handling HTTP requests. This function will connect to the postgres database and retrieve all data from users table and return the result in JSON format. The code for this function is shown in Figure 31, line 59 – 82 :



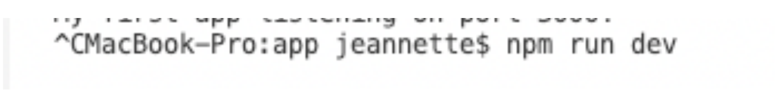
```javascript
59  app.get('/users', async (req,res) => {
60      try{
61          let results;
62          const pool = new pg.Pool(config);
63          const client = await pool.connect();
64          const q = 'select * from users;'
65          await client.query(q, (err, results) => {
66            if (err) {
67              console.log(err.stack)
68              errors = err.stack.split(" at ");
69              res.json({ message:'Sorry something went wrong! The data has not been processed ' + errors[0]});
70            } else {
71              client.release();
72             // console.log(results); //
73              data = results.rows;
74              count = results.rows.length;
75              res.json({ results:data, rows:count });
76            }
77          });
78
79      }catch(e){
80          console.log(e);
81      }
82  });
83
84
85  app.listen(port, () => {
86      console.log(`My first app listening on port ${port}!`)
87  });
```

Figure 31. The code for users GET route

Code explained: create a Get route for path called users with an asynchronous callback function. The asynchronous function is indicated by using the async (MDN) keyword - line 59. We put the code inside a try block (MDN) in case there was any errors.

If there are errors, the catch block (MDN) will handle them appropriately. Create a variable to store the results – line 61. Next, create a pool (source) object using the config details – line 62, create a client from the pool object – line 63 and use the client to execute the query – line 65. Note that I have created the query string in line 64. The `query` method takes a callback function – line 65 and handle errors – line 66 – 69 and results – line 71 – 75 appropriately. Note that the results will send back in JSON format – line 75.

6) To test the code, run the development server on a Terminal window:



```
^CMacBook-Pro:app jeannette$ npm run dev
```

Figure 32. Run local development server

Make sure that you are in the project root directory

7) On Chrome type below URL in your address bar:

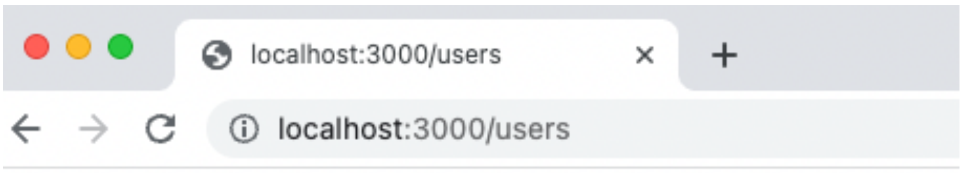
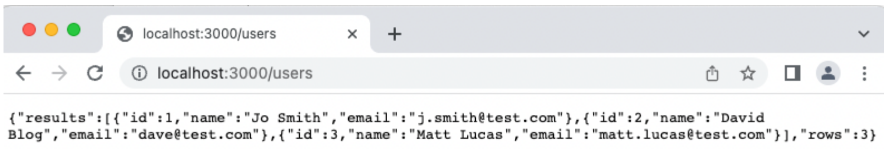


Figure 33. Google Chrome for testing

8) Figure 34 below is my results for you to compare:



```
{"results":[{"id":1,"name":"Jo Smith","email":"j.smith@test.com"},{"id":2,"name":"David
Blog","email":"dave@test.com"},{"id":3,"name":"Matt Lucas","email":"matt.lucas@test.com"}],"rows":3}
```

Figure 34. The results are rendered in JSON format.

9) Your next task is to create a client script called `users.js` and render the results on `hello.html` page. It would be nice if there is a button on this page so that the data is only rendered upon the button being clicked – as shown in Figure 35, and results are shown in Figure 36. Implement a separate button to clear the results – as shown in Figure 37.

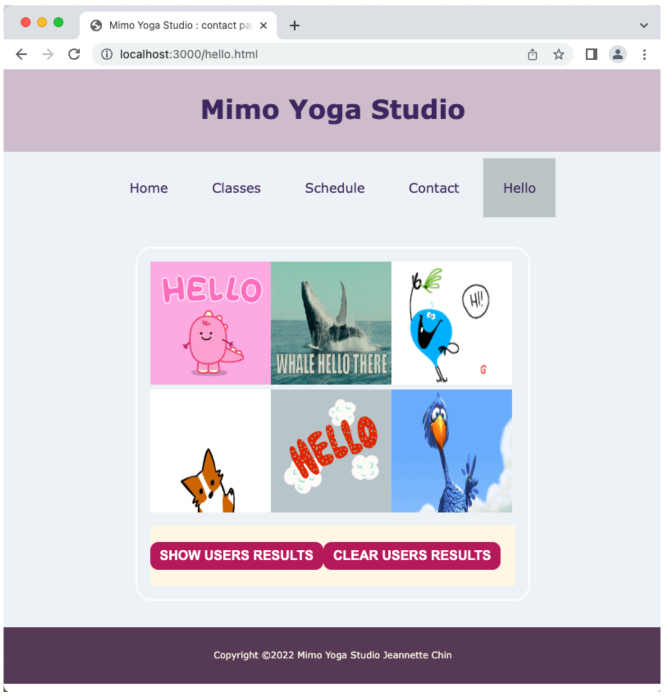
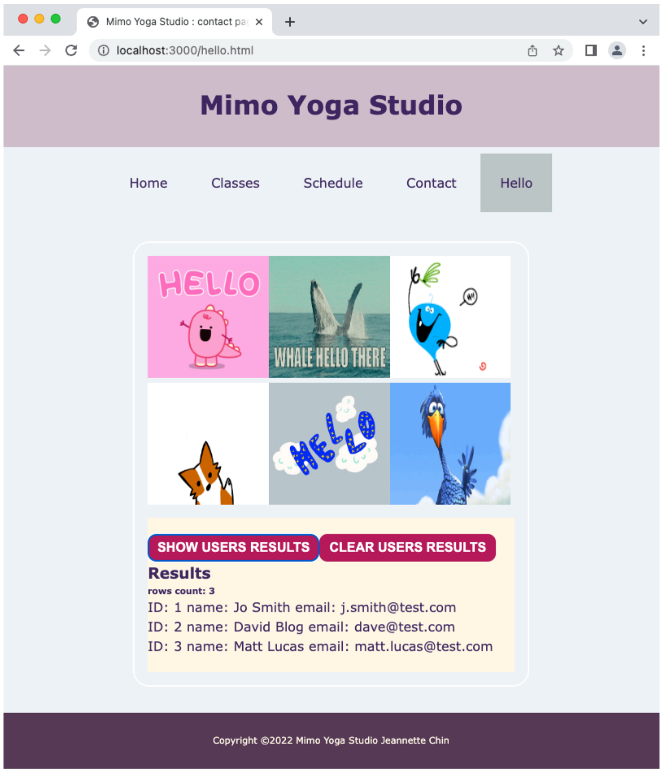

Mimo Yoga Studio : contact pa
localhost:3000/hello.html
Mimo Yoga Studio
Home
Classes
Schedule
Contact
Hello
SHOW USERS RESULTS
CLEAR USERS RESULTS
Copyright ©2022 Mimo Yoga Studio Jeannette Chin

Figure 35. Two additional buttons on hello.html page


Mimo Yoga Studio : contact pa
localhost:3000/hello.html
Mimo Yoga Studio
Home
Classes
Schedule
Contact
Hello
SHOW USERS RESULTS
CLEAR USERS RESULTS

```
Results
rows count: 3
ID: 1 name: Jo Smith email: j.smith@test.com
ID: 2 name: David Blog email: dave@test.com
ID: 3 name: Matt Lucas email: matt.lucas@test.com
```

Copyright ©2022 Mimo Yoga Studio Jeannette Chin

Figure 36. Users' results show upon the "Show Users Results" button is being clicked

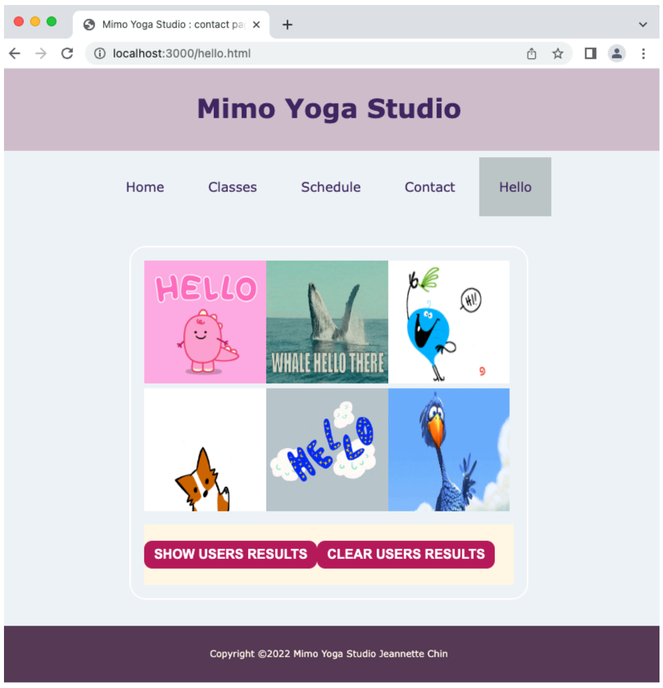


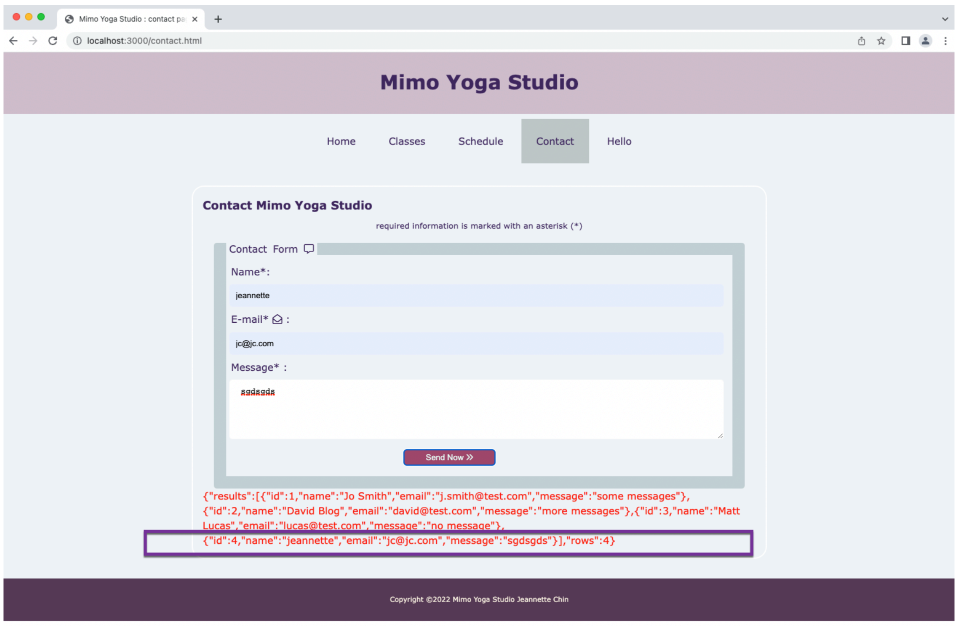
Figure 37. Results are clear upon the "Clear users results" button is being clicked.

v. Next, using what you have learned so far, create a table that can store the data in the contact form (i.e. name, email and message). [**Hint:** drop the existing table and create a new one with appropriate fields]. Make sure that this is done before moving on to the next task.

   The remaining task for you to do is to add the form data to this table. **Hint:** You will need to use SQL insert statement (source) and JS template literals (MDN) for the following task.

vi. Write some code to insert the data in this new table when the "Send Now" button is clicked.
   a. Your client code should be named as `adduser.js`
   b. A new POST route with a path called "`adduser`" should be implemented in your `app.js`. Make sure that data is sanitised and validated to prevent injection vulnerability (OWASP)

Make sure that the data is inserted correctly in the database. My outputs are shown in Figure 38 and 41 for you to compare.



Contact Mimo Yoga Studio

required information is marked with an asterisk (*)

Contact Form

Name*: jeannette

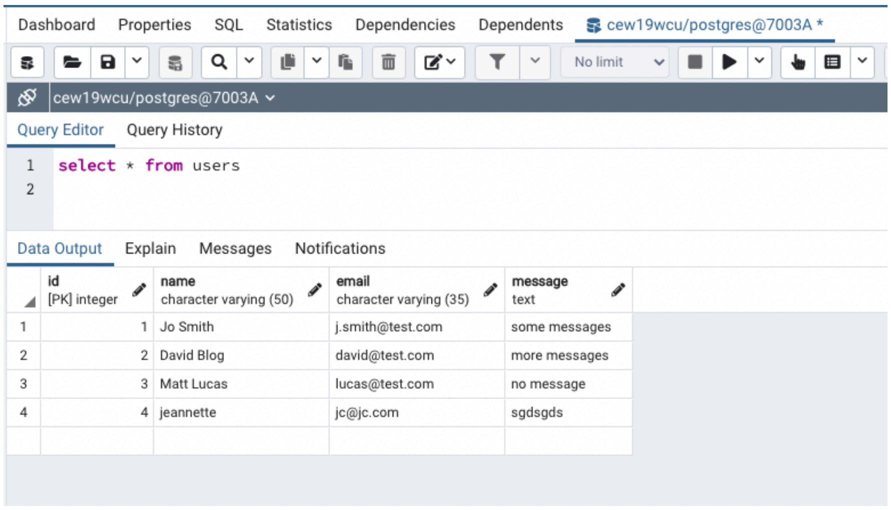
E-mail* : jc@jc.com

Message* : sgdsgds

```
{"results":[{"id":1,"name":"Jo Smith","email":"j.smith@test.com","message":"some messages"},
{"id":2,"name":"David Blog","email":"david@test.com","message":"more messages"},{"id":3,"name":"Matt
Lucas","email":"lucas@test.com","message":"no message"},
{"id":4,"name":"jeannette","email":"jc@jc.com","message":"sgdsgds"}],"rows":4}
```

Copyright ©2022 Mimo Yoga Studio Jeannette Chin

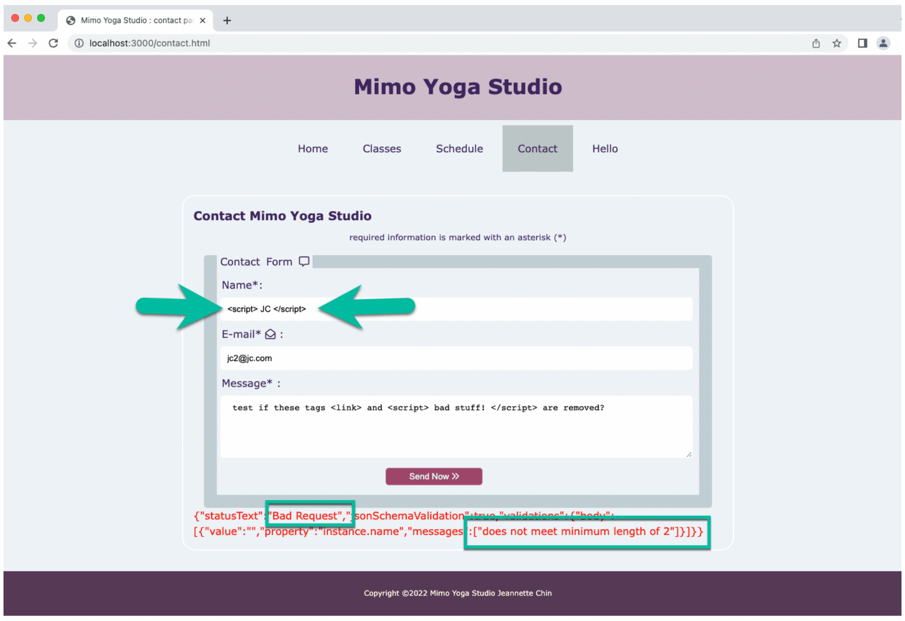
Figure 38. The results of adding data to database



```
select * from users
```

	id [PK] integer	name character varying (50)	email character varying (35)	message text
1	1	Jo Smith	j.smith@test.com	some messages
2	2	David Blog	david@test.com	more messages
3	3	Matt Lucas	lucas@test.com	no message
4	4	jeannette	jc@jc.com	sgdsgds

Figure 39. The database results


Mimo Yoga Studio : contact pa
localhost:3000/contact.html
Mimo Yoga Studio
Home
Classes
Schedule
Contact
Hello
Send Now »
Copyright ©2022 Mimo Yoga Studio Jeannette Chin

Contact Mimo Yoga Studio

required information is marked with an asterisk (*)

Contact Form

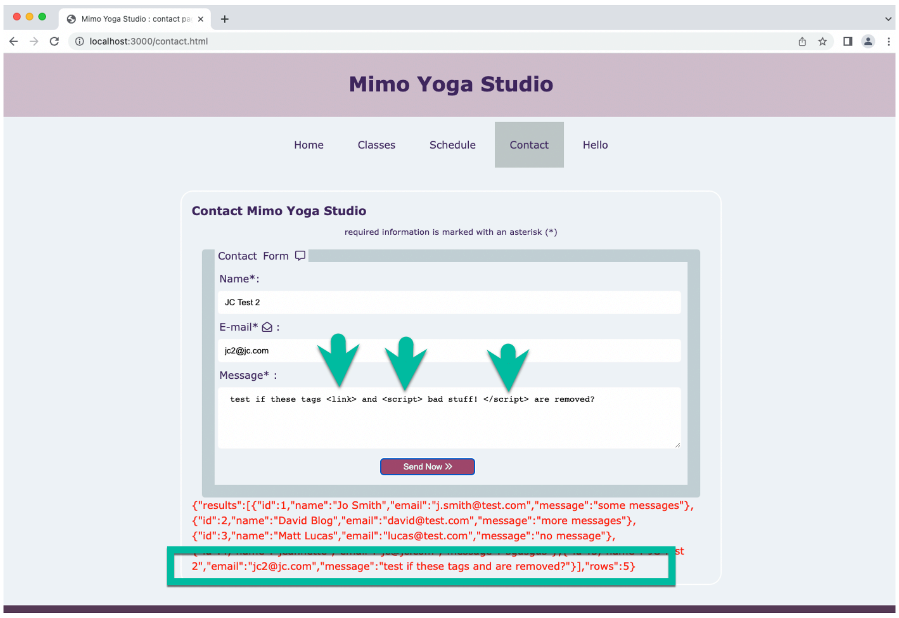
Name*: <script> JC </script>

E-mail* : jc2@jc.com

Message* : test if these tags <link> and <script> bad stuff! </script> are removed?

```
{"statusText":"Bad Request","jsonSchemaValidation":true,"validations":{"body":
[{"value":"","property":"instance.name","messages":["does not meet minimum length of 2"]}]}}
```

Figure 40. Test malicious input 1



Contact Mimo Yoga Studio

required information is marked with an asterisk (*)

Contact Form

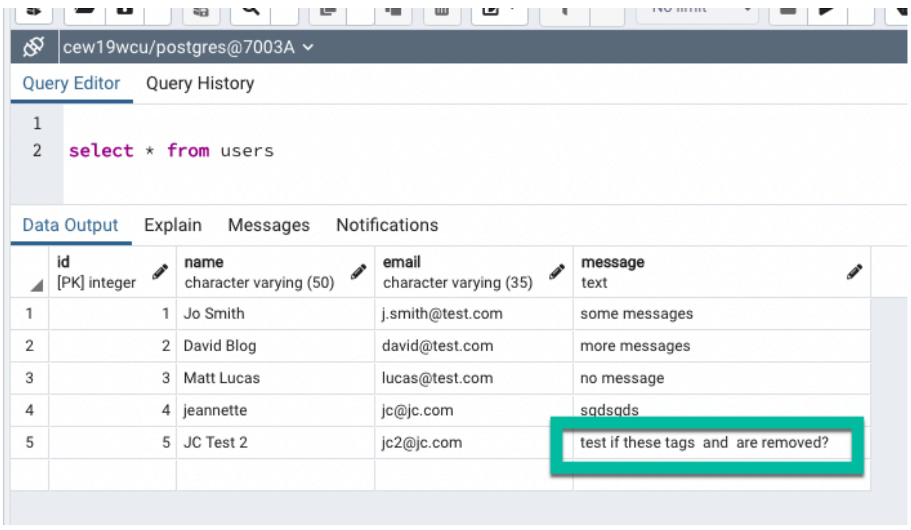
Name*: JC Test 2

E-mail* : jc2@jc.com

Message* : test if these tags <link> and <script> bad stuff! </script> are removed?

```
{"results":[{"id":1,"name":"Jo Smith","email":"j.smith@test.com","message":"some messages"},
{"id":2,"name":"David Blog","email":"david@test.com","message":"more messages"},
{"id":3,"name":"Matt Lucas","email":"lucas@test.com","message":"no message"},
[illegible]st
2","email":"jc2@jc.com","message":"test if these tags and are removed?"}],"rows":5}
```

Figure 41. Test malicious input 2


cew19wcu/postgres@7003A

Query Editor | Query History

```
select * from users
```

Data Output | Explain | Messages | Notifications

	id [PK] integer	name character varying (50)	email character varying (35)	message text
1	1	Jo Smith	j.smith@test.com	some messages
2	2	David Blog	david@test.com	more messages
3	3	Matt Lucas	lucas@test.com	no message
4	4	jeannette	jc@jc.com	sgdsgds
5	5	JC Test 2	jc2@jc.com	test if these tags and are removed?

Figure 42. Verify the data in Postgres database