
Exercise. Todo (part 2) – Creating backend

On the second part a simple backend is created for the Todo web app. Make sure, that you have Node installed (in this example version 18.x is used, but 20.x LTS should work)

Key learnings for this part are:

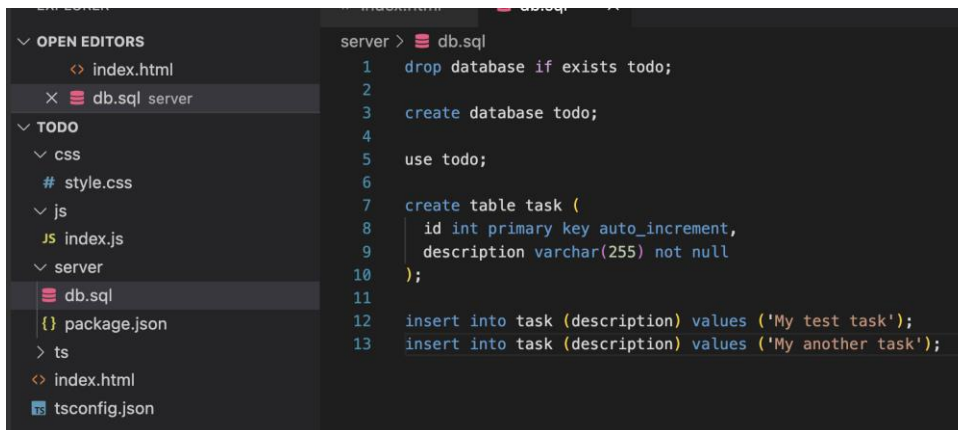
- Create PostgreSQL database with some content
- Create simple NodeJS app using JavaScript and Express
- Basic database programming using pg library
- Testing backend with REST Client

Start by creating a folder under name server to the project. Create an empty NodeJS project by running following command (*npm init -y*). Make sure that you run the command under server folder on terminal. Package.json file is generated under server folder.

```
● jjuntune@vpn-10-2-96-78 todo % cd server
○ jjuntune@vpn-10-2-96-78 server % npm init -y
```

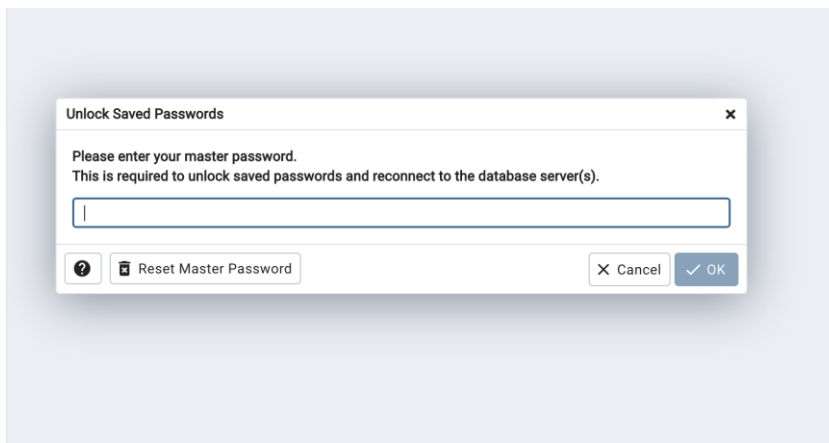
[Download](#) and install PostgreSQL which is an open-source relational database. Use default settings for installation. Remember password, that you need to provide during installation, since it is required later. Before finishing installation StackBuilder tool is not required, so you can uncheck the box before finishing installation.

Add SQL file for creating table and initial data.

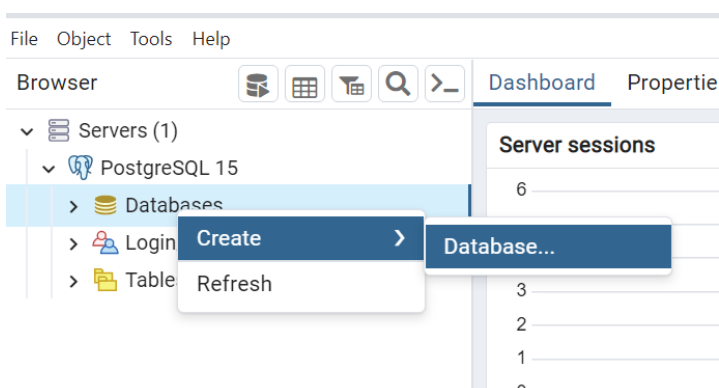


```
server > db.sql
1 drop database if exists todo;
2
3 create database todo;
4
5 use todo;
6
7 create table task (
8   id int primary key auto_increment,
9   description varchar(255) not null
10 );
11
12 insert into task (description) values ('My test task');
13 insert into task (description) values ('My another task');
```

Open pgAdmin4. Login using password created earlier during installation.



Extend Servers and select PostgreSQL 15 (or the version that you have installed). Under Databases, right click and select Create -> Database...



Provide a name for the database (todo). The owner is the default user (postgres).

Create - Database

General Definition Security Parameters Advanced SQL

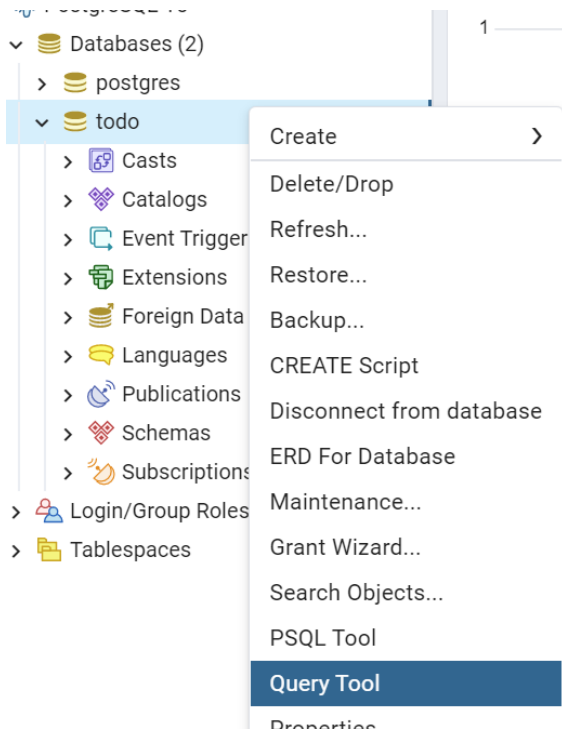
Database: todo

Owner: postgres

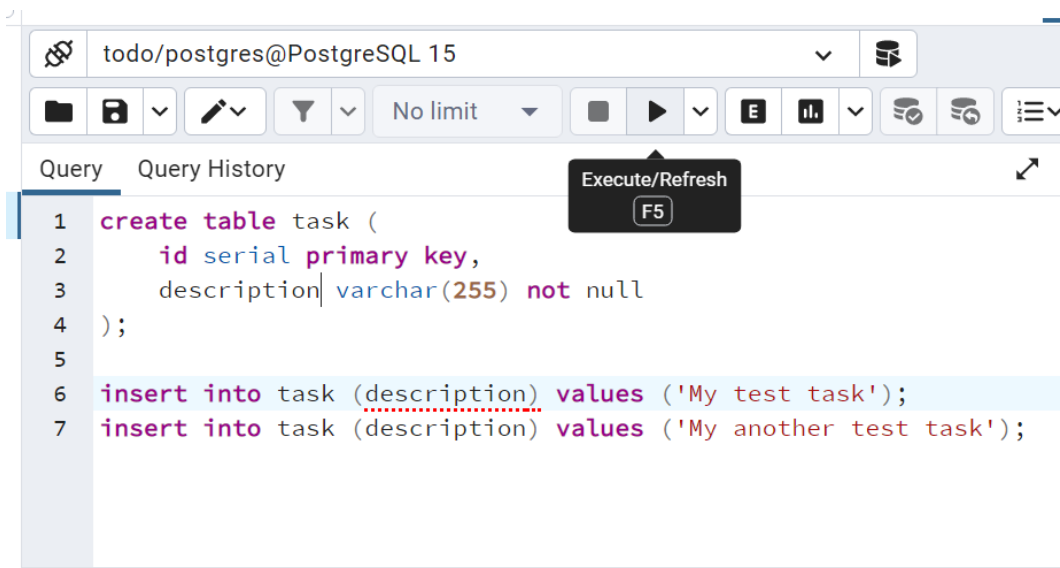
Comment:

Close Reset Save

Open Query Tool.



Copy content from the SQL file that was created (and saved) earlier, paste it to the window, and execute commands. This should create a table task for the Todo database.



NodeJS server need some libraries. The following libraries are used.

- **Cors** for enabling clients to connect to this service without any Cors restrictions.
- **Express** is used for creating HTTP endpoints (get, post, etc.).
- **Nodemon** is used to watch changes in code and restart the backend server automatically when changes occur.
- **PG** is used to access the Postgres database from the NodeJS app.

```
PS C:\Users\Admin\Development\Typescript\todo\server> npm i express cors nodemon pg
```

Define scripts that are used to launch the NodeJS server. DevStart script launches the server application using nodemon, which automatically restarts the server app when there are changes made to the source code (and there is no need to stop/start the server manually repeatedly). Do the following modification to package.json under the server folder.

```
Debug  
"scripts": {  
  "test": "echo \"Error: no test specified\" && exit 1",  
  "devStart": "nodemon index.js"  
},  
"keywords": [],
```

▸

Create an `index.js` file under the `server` folder. Create a script, that starts the NodeJS server and returns a simple JSON response. Express and Cors libraries are used. Express helps to define endpoints (in this example root address `/`) and cors enables testing this backend service from different origins (e.g. client might be using port 5500 and the backend is running on port 3001).

```
const express = require('express')
const cors = require('cors')

const app = express()
app.use(cors())
const port = 3001

app.get("/", (req, res) => {
  res.status(200).json({result: 'success'})
})

app.listen(port)
```

Test that you can call your server using the address `http://localhost:3001`. Make sure that you have started the server using `npm run devStart` command. Occasionally when making a programming mistake server might shut down, so make always sure, that the server is running before testing. Remember also, that whenever you start working on this project, you need to remember to start backend service.

```
PS C:\Users\Admin\Development\Typescript\todo\server> npm run devStart
```

Add the require definition to use the `pg` library.

```
const express = require('express')
const cors = require('cors')
const { Pool } = require('pg')
```

Modify the `get` function to open the database connection and retrieve data from the database. A function for opening the database is implemented, so it can be called from

multiple other functions (as illustrated later). Provide the required credentials and information to open the database connection.

Connection pool is used to open database connection and query method is used to execute SQL statements towards opened database. The query returns possible errors and results. According to returned values either error or result is returned as JSON. Also, the appropriate HTTP status code (200 OK or 500 Internal Server Error) is returned. In this example, port 5432 is used (which is commonly used in Windows OS). But it might be something different, for example, 5435 in Mac OS.

```
app.get("/", (req, res) => {  
  const pool = openDb()  
  
  pool.query('SELECT * FROM task', (error, result) => {  
    if ( error ) {  
      res.status(500).json({ error: error.message })  
    }  
    res.status(200).json(result.rows)  
  })  
})
```

```
const openDb = (): Pool => {  
  const pool: Pool = new Pool ({  
    user: 'postgres',  
    host: 'localhost',  
    database: 'todo',  
    password: 'root',  
    port: 5432  
  })  
  return pool  
}
```

Test out the get using a browser. Data retrieved from the database should be displayed as JSON. Make sure that Postgres is running.



```
[{"id":1,"description":"My test task"}, {"id":2,"description":"My another test task"}]
```

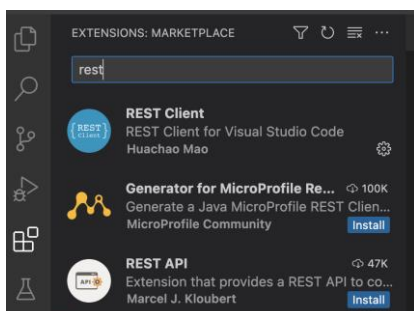
Add following use statement to index.js, which allows reading posted values from the client as JSON.

```
app.use(cors())  
app.use(express.json())  
  
const port = 3001
```

Implement a post handler, which is used to receive value(s) from the client and execute insert into statement into the database. SQL contains some parameters (\$1) which are provided as an array when calling the query method. Return defines that data inserted into the database will be returned as a result (and we can read, for example, the id that was generated for the new record in the database).

```
app.post("/new", (req, res) => {  
  const pool = openDb()  
  
  pool.query('insert into task (description) values ($1) returning *',  
    [req.body.description],  
    (error, result) => {  
      if (error) {  
        res.status(500).json({error: error.message})  
      } else {  
        res.status(200).json({id : result.rows[0].id})  
      }  
    })  
})
```

Browser is good for testing simple GET HTTP requests. A more sophisticated tool is, for example, REST Client. Open extensions on Visual Studio Code and install it.



Create client.rest file under the server folder with the following content. The first piece

of code is executing HTTP GET request (basically the same thing done with the browser earlier). The second call is POST which sends JSON to the server (description for task that is added to the database). It is good practice to test the backend and make sure that it works before developing the front-end. Press Send Request links to see what server responds.

```
### Get tasks
Send Request
GET http://localhost:3001

### Add new task
Send Request
POST http://localhost:3001/new HTTP/1.1
Content-Type: application/json

{
  "description": "Test from REST Client"
}

1 HTTP/1.1 200 OK
2 X-Powered-By: Express
3 Access-Control-Allow-Origin: *
4 Content-Type: application/json; charset=utf-8
5 Content-Length: 9
6 ETag: W/"9-R0vkHEm9hjIzQAeCBxLFT7Mb/uQ"
7 Date: Mon, 27 Feb 2023 15:03:40 GMT
8 Connection: close
9
10 {
11   "id": 12
12 }
```