Project presentation

Content:

Create relational database as backend #13

Install Python packages in the extension

I tried to get a more recent version of the Docker container of timescaleDB-PostgreSQL above, but it does not work with timescaledb:latest-pg11 till timescaledb:latest-pg13, since the Dockerfile throws the error:

```
postgresql-plpython3 (no such package): required by: .plpython3-deps-20210819.182405 [postgresql-plpy
thon3]
```

It seems as if only PostgreSQL 10 gives you the chance to use plpython with that timescaleDB container.

To check whether timescale on Docker is the only way to go, I installed PostgreSQL 10 on Windows with the official EDB installer and on top of that, using Stack Builder to add pl/python to the EDB installation, which is recommended. When this still did not work, I copied the pyhton37.dll into the Windows\System32 directory as a known trick, but it still did not work. That is why it seems that this timescaleDB-PostgreSQL container is perhaps the only configuration on the net with a working pl/python extension. It was pure luck to find out about the timescaleDB-PostgreSQL container, that is why this is disappointing PostgreSQL service. Funny enough, timescaleDB does not offer plpython on the web server which could mean that plpython is just too insecure to be allowed on production systems and therefore was ignored by the recent developments.

That is why I must take the 2016 PostgreSQL 10 on a timescaleDB only to test plpython. Very strange, and likely not the best way to go. It all rather hints at Spark to replace PostgreSQL, as planned in issue #17.

After a full day invested into installing additional basic packages like pandas in the Alpine Docker container of PostgreSQL10 and timescale 0.9.0 (FROM timescale/timescaledb:0.9.0-pg10) I had to find out that apk (Alpine form of apt) does not support (well enough or not at all) basic packages in exactly this old Python 3.6 version, see Installing pandas in docker Alpine. I have installed Poetry to get the dependencies right, but it did not work.

After having tried getting Python extension and the packages to run on Windows and on an outdated Alpine Docker container, I have now eventually succeeded in installing PostgreSQL and plpython3u on Linux (WSL2).

There is an official guide for Linux installations at PostgreSQL Downloads and PostgreSQL Wiki.

This is how to install it:

```
# Create the file repository configuration:
sudo sh -c 'echo "deb <http://apt.postgresql.org/pub/repos/apt> $(lsb_release -cs)-pgdg main" > /etc/
apt/sources.list.d/pgdg.list'

# Import the repository signing key:
wget --quiet -O - <https://www.postgresql.org/media/keys/ACCC4CF8.asc> | sudo apt-key add -

# Update the package lists:
sudo apt-get update

# Install the latest version of PostgreSQL.
# If you want a specific version, use 'postgresql-12' or similar instead of 'postgresql':
sudo apt-get -y install postgresql
```

After this, install the extension, but check before whether you have two postgresql versions installed by running

```
service postgresql start
```

If you see two or more, as I had versions 12 and 13, consider either deleting the unneeded or changing the config and settings by following this link.

Install the plpython3u extension following PostgreSQL: how to install plpythonu extension:

```
sudo apt-cache search ".*plpython3.*"
sudo apt-get install postgresql-contrib postgresql-plpython3-13
```

Now change to the postgres user:

```
sudo su postgres
```

If you have two postgresql versions installed, you need to run psql with the right port.
If your 13 version is at port 5433, run

```
psql --5433
```

If you only have one version installed, run

```
psql
```

After this, follow the typical testing of plpython3u by creating the return_version() function of above and checking its results.

Half a day invested into getting an import of Python packages done for a plpython3u stored procedure. No way up to now. The installation of pandas in Python 3.8.2 has no effect on the Python version 3.8.10 reported by PostgreSQL, the kmeans test function still asks for pandas. I did not understand how to use the solution of “Module not found” when importing a Python package within a plpython3u procedure.

This is done now. The easy mistake I made was to install the packages without sudo in front. We are now able to use the full range of python in stored procedures on database level.

Next steps are at #29.

Use relational database for clustering#29

Starting point, see #13:

- relational database
- working plpython3u extension
- Python packages can be installed.

I changed the kmeans test function so that it returns not a pickle dump, but a table (a merger of the df and the new column for the kmeans cluster).

```
CREATE OR replace FUNCTION kmeans3(input_table text, columns text[], clus_num int) RETURNS table(lon
 float, lat float, k float) AS

$$

from pandas import DataFrame
from sklearn.cluster import KMeans
#from pickle import dumps

all_columns = ",".join(columns)
if all_columns == "":
    all_columns = "*"

rv = plpy.execute('SELECT %s FROM %s;' % (all_columns, plpy.quote_ident(input_table)))



frame = []
```

```
for i in rv:
    frame.append(i)
#df = DataFrame(frame).convert_objects(convert_numeric =True)
#df = pandas.to_numeric(DataFrame(frame))
df = DataFrame(frame).astype(float)
print(df.shape)
kmeans = KMeans(n_clusters=clus_num, random_state=0).fit(df._get_numeric_data())
df['kmeans'] = kmeans.labels_ #.astype(float)
return df.values

$$ LANGUAGE plpython3u;
```

You can ask for the results with:

```
SELECT * FROM kmeans3('stokes', ARRAY['lon', 'lat'],3);
        lon        |        lat         | k
-------------------+--------------------+---
 5.171320915222168 | 43.288516998291016 | 0
 4.982897758483887 |  43.29656219482422 | 2
 4.962841033935547 |  43.29465103149414 | 2
 5.100956439971924 |  43.28042221069336 | 0
 5.134312629699707 |  43.29485321044922 | 0
 5.279866695404053 | 43.274662017822266 | 1
 5.095328330993652 |  43.30269241333008 | 0
 5.301522731781006 |  43.29117202758789 | 1
 5.205143928527832 |  43.30012512207031 | 1
(9 rows)
```

Strangely, it seems necessary to have k column as float in the return value, although there are clearly just integers in it. Typecast to int was not accepted. But it must be possible to export other data types to the same table. Small TODO.

To save the table result to a postgres table directly, either create the table in advance and insert:

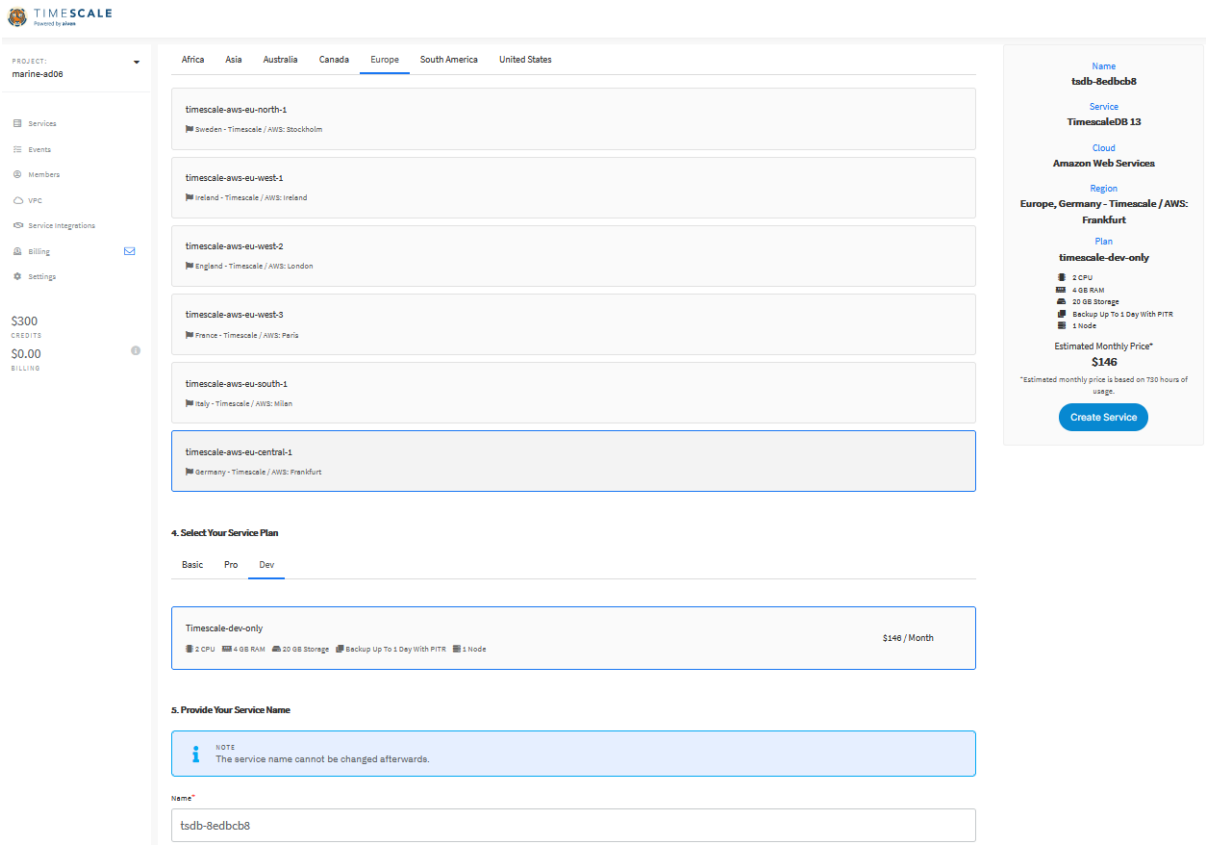
```
create table tab_kmeans1(lon float, lat float, k float);
insert * into tab_kmeans1 SELECT * FROM kmeans3('stokes', ARRAY['lon', 'lat'],3);
```

or create a new table from the output table:

```
select * into tab_kmeans1 FROM kmeans3('stokes', ARRAY['lon', 'lat'],3);
```

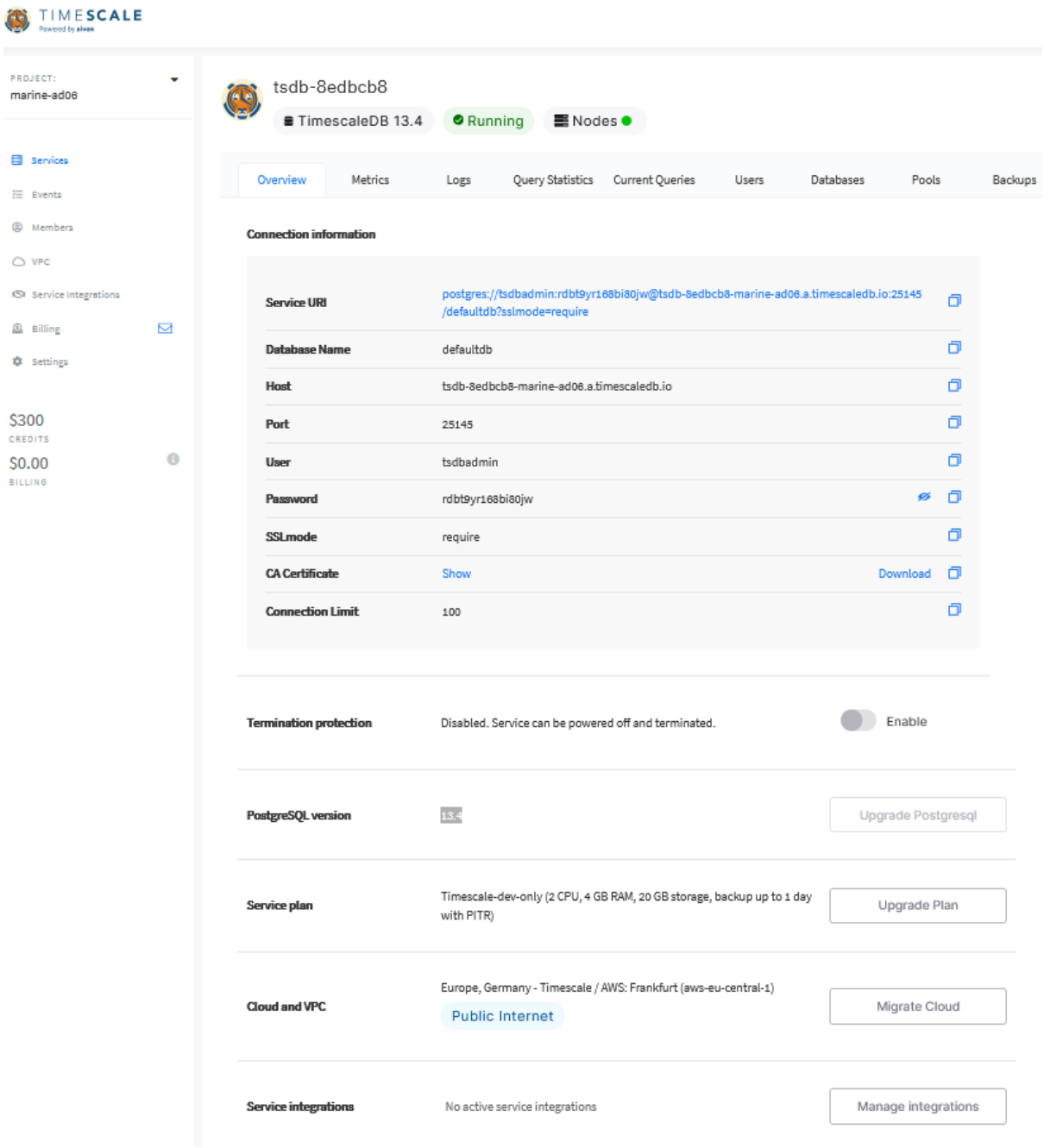
Webserver as backend for the mobile app#38

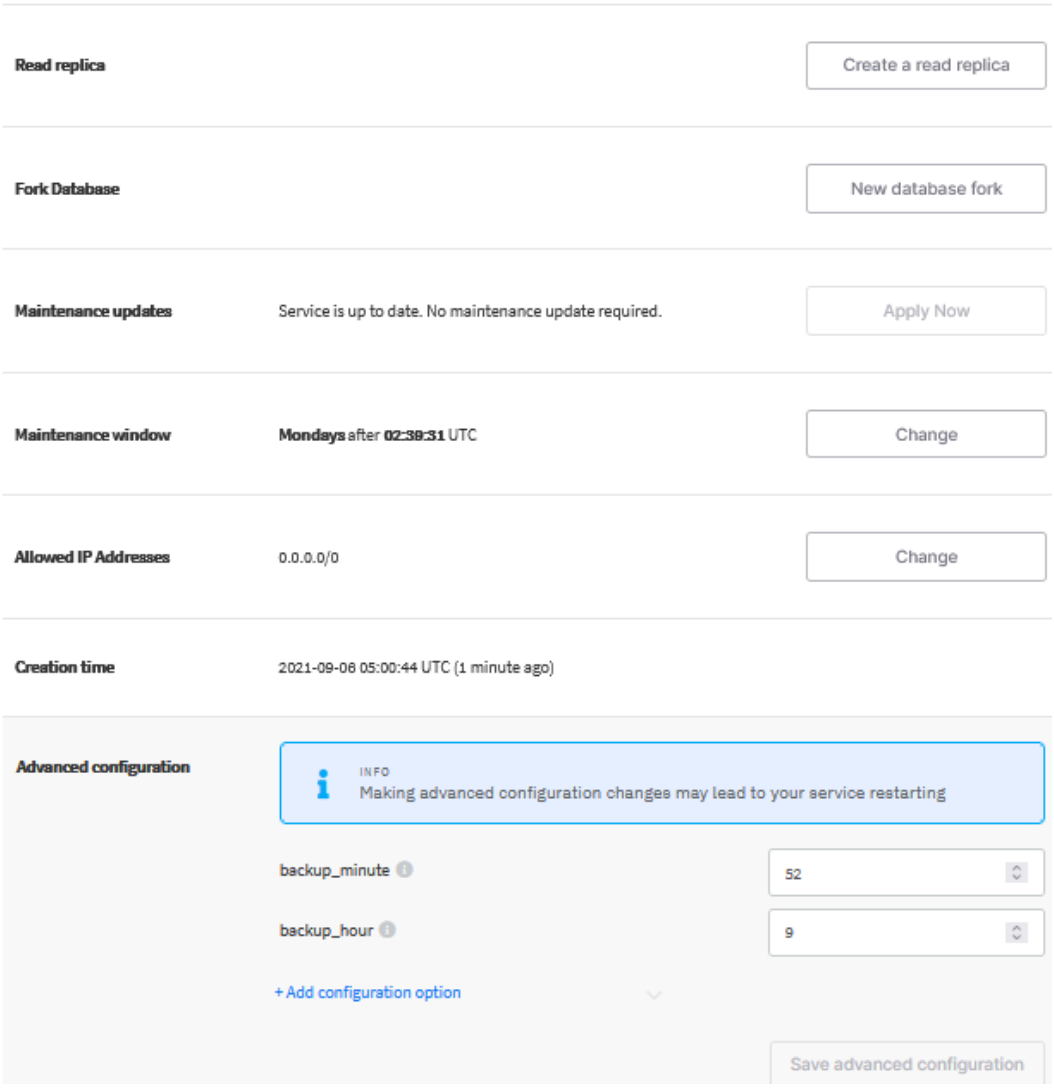
Created a new Webserver and the menu on TimeScale has changed, therefore some new screenshots:



Details:

- Service name: tsdb-8edbcb8
  - Cloud: timescale-aws-eu-central-1
  - Plan: timescale-dev-only


TIMESCALE
PROJECT:
marine-ad06
Services
Events
Members
VPC
Service Integrations
Billing
Settings
$300
CREDITS
$0.00
BILLING
tsdb-8edbcb8
TimescaleDB 13.4
Running
Nodes
Overview
Metrics
Logs
Query Statistics
Current Queries
Users
Databases
Pools
Backups
Connection information
Service URI
postgres://tsdbadmin:rdbt9yr168bi80jw@tsdb-8edbcb8-marine-ad06.a.timescaledb.io:25145/defaultdb?sslmode=require
Database Name
defaultdb
Host
tsdb-8edbcb8-marine-ad06.a.timescaledb.io
Port
25145
User
tsdbadmin
Password
rdbt9yr168bi80jw
SSLmode
require
CA Certificate
Show
Download
Connection Limit
100
Termination protection
Disabled. Service can be powered off and terminated.
Enable
PostgreSQL version
13.4
Upgrade Postgresql
Service plan
Timescale-dev-only (2 CPU, 4 GB RAM, 20 GB storage, backup up to 1 day with PITR)
Upgrade Plan
Cloud and VPC
Europe, Germany - Timescale / AWS: Frankfurt (aws-eu-central-1)
Public Internet
Migrate Cloud
Service integrations
No active service integrations
Manage integrations



Connection information

Service URI: postgres://tsdbadmin:rdbt9yr168bi80jw@tsdb-8edbcb8-marine-ad06.a.timescaledb.io:25145/defaultdb?sslmode=require

Database Name: defaultdb
Host: tsdb-8edbcb8-marine-ad06.a.timescaledb.io
Port: 25145
User: tsdbadmin
Password: rdbt9yr168bi80jw
SSLmode: require
CA Certificate: Download
Connection Limit: 100

CA Certificate, if needed:
filename: ca.pem
-----BEGIN CERTIFICATE-----
MIIEQTCCAqmgAwIBAgIUPVry8oOWmxx4eHNWRAlKvdPPMSkwDQYJKoZIhvcNAQEM
BQAwOjE4MDYGA1UEAwwvZDk2ZDM4MzQtNmFmMS00NmFlLWEyMjUtZGMwMDE1MDU0

YjJlIFByb2plY3QgQ0EwHhcNMjEwOTA2MDQyMDA3WhcNMzEwOTA0MDQyMDA3WjA6
MTgwNgYDVQQDDC9kOTZkMzgzNC02YWYxLTQ2YWUtYTIyNS1kYzAwMTUwNTRiMmUg
UHJvamVjdCBDQTCCAaIwDQYJKoZIhvcNAQEBBQADggGPADCCAYoCggGBALUjayZk
aZ5yjj3vjbtyPe8okGHniMwpll/gyTuhuQcqu2rxcZPDRlkgAjgYYVwmx+pf8DVY
A2xNB8Lxol+Kt2kBHW24/dBzD3UOqmDcwSgufwYtSB1Ql9vaWWtCPgod1uf31mSv
BW+XyoUYY/KRrK9+GslQTn70xDL5vxFumqLe9MkAS28NdPk2Gl7aNHKrEHAMCsl9
MOEgbe6tuKFbfYjMUrkGNaYXSpEok92gvKRcYhoK+ocgS8aQEuSS6ah2bqnF8Cf0
rYyXIC3CUdkVL1/mDTTVgaJb+2k2HisIZ86/oauuYepE5zhMaVp8zrfTz2LcB9Wv
878EYO0zwiggxjI7AHMiqooZaxPlhi7vvTWaqGJkLhK0NY0sthlOqEX14MFQJKO4
9m4WH1zcntGIxOTPV3vzsOIq86SWSctkd3HipM2CXgbbSaGbRQO5SHTqomhJu7p+
TYIsL1bW17ddG1BDLXxgPbOYpX+XuxM8Z6CPUzLxv9+aSQoVGTdfFFNGEwIDAQAB
oz8wPTAdBgNVHQ4EFgQUqOut8Uppgng2BLIekQ8WQOtFPc8wDwYDVR0TBAgwBgEB
/wIBADALBgNVHQ8EBAMCAQYwDQYJKoZIhvcNAQEMBQADggGBAJrnDwKGvzV3FKd7
+OylOI+M/PI4MnYKDtyECTlOVmIIJzJ6IrbqhZ04gH/uGs4xGpAxmAtSwx6s9AdR
U8wIkhmcb9ytWFC/hyoK7v1EDEJWidxhTRtooE01y/mdZYiTFKOL9L0f/8H0tlGf
V937TBZjmp4jyINMN/urR2LSiNYYuJo2ZgiAZv8Lw8TfKbolgzqstRCUn2G1MYFg
HKTehiiBMoOaixSZ3K1Kq3o7JBwAT4r1eo2rz4FDNMV37vP52K2BpA/h3259bBTE
ig7lTNeb2bxUBR2gmDj6uy0UTmErfv9HBb0mtqEq3H4y1hG0vnfXH6jPf7U5LeEq
MAgLOX3gFVvBjM3eTc6LZugniyNK99114CSU8vKW08A6NNRHFLIFK7+cRjv76wSo
KLXEfYabqoagaw/2EhrC2UfpBfdpMYAf/FQCaRFB+lPRrm/ds6gHbh94g03UpUW1
5oFysnUa2wmcqyqv86bVcPfQfupJ383rXxtRwOPcVOEdP4eSdg==
-----END CERTIFICATE-----

Since SSL is required, the above certificate needs to be used to connect to the database. I am still trying to figure that out on Windows.

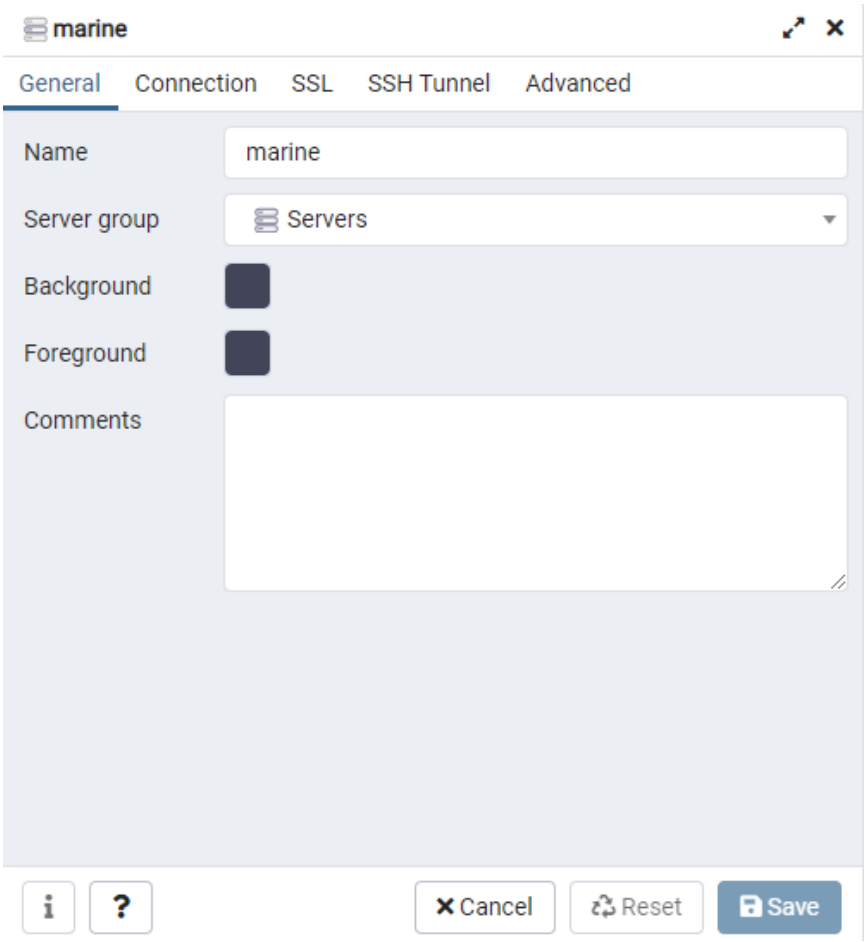
```
Server [localhost]: postgres://tsdbadmin:rdbt9yr168bi80jw@tsdb-8edbcb8-marine-ad06.a.timescaledb.io:25145/defaultdb?sslm
ode=require
Database [postgres]: defaultdb
Port [5432]: 25145
Username [postgres]: tsdbadmin
psql: Fehler: konnte Hostnamen »postgres://tsdbadmin:rdbt9yr168bi80jw@tsdb-8edbcb8-marine-ad06.a.timescaledb.io:25145/de
faultdb?sslmode=require« nicht in Adresse übersetzen: Unknown server error
Drücken Sie eine beliebige Taste . . .
```

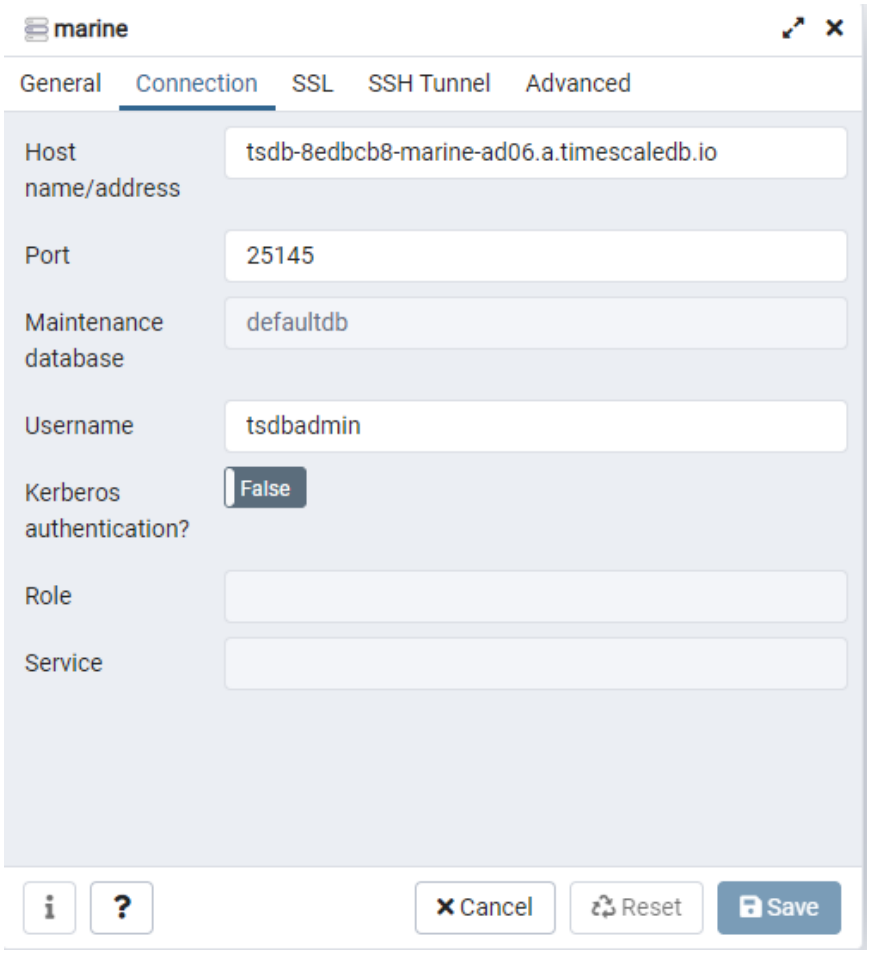
It seems to be rather supported on Linux only.

The connection URL and other parameters as well as the certificate are now available for everyone, see above. The database is still empty, though, I first need to find out how to connect. (TODO) DONE

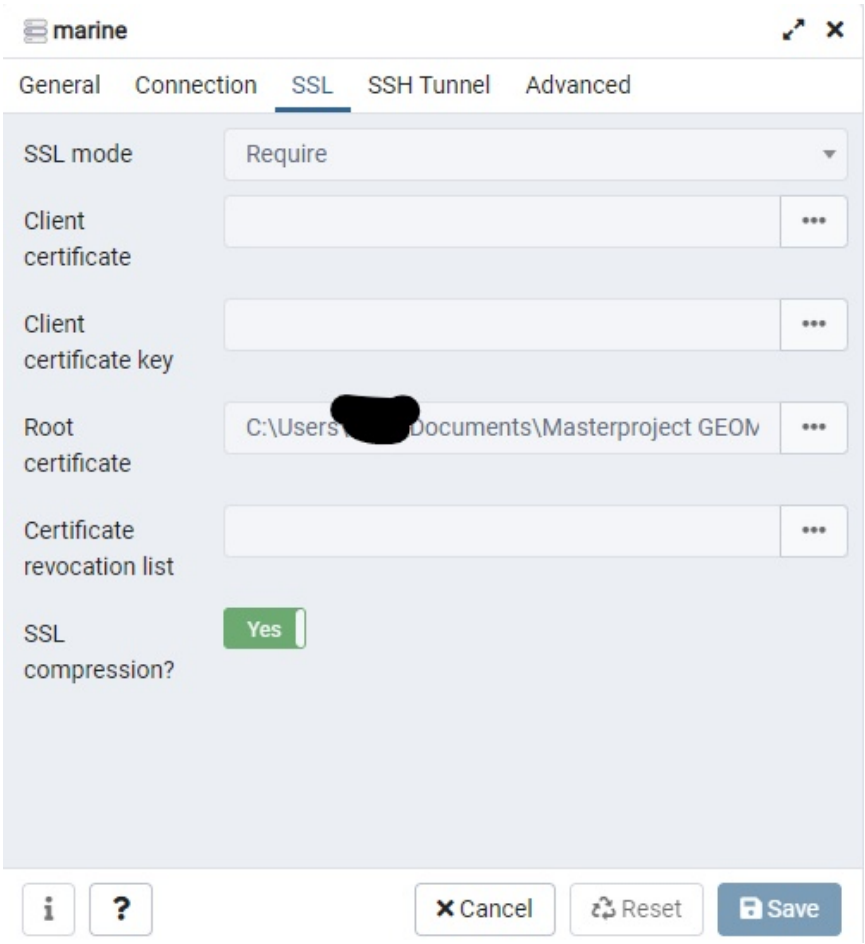
Connection with root certificate for SSL encryption is now DONE

On Windows, with pgAdmin 4 v5:

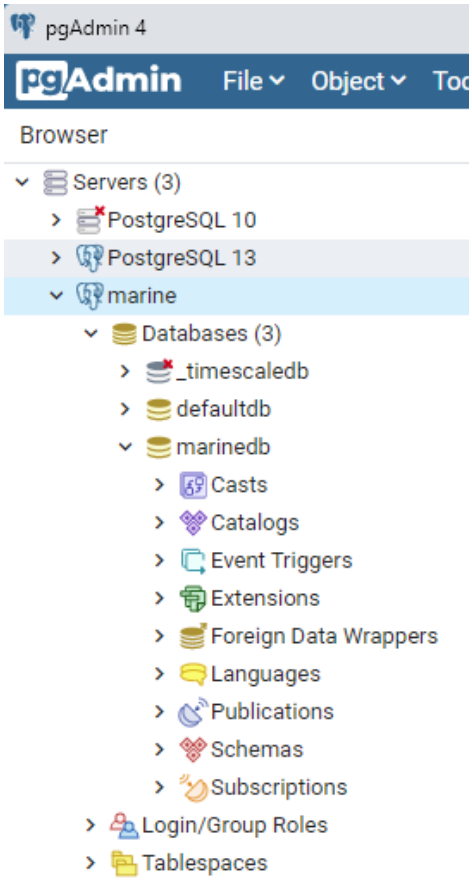

marine
General
Connection
SSL
SSH Tunnel
Advanced
Name
marine
Server group
Servers
Background
Foreground
Comments
Cancel
Reset
Save



Choose the "ca.pem" certificate:


marine
General
Connection
SSL
SSH Tunnel
Advanced
SSL mode
Require
Client certificate
Client certificate key
Root certificate
C:\Users
Documents\Masterproject GEOM
Certificate revocation list
SSL compression?
Yes
Cancel
Reset
Save

Result:



On Windows, with psql

```
Server [localhost]:  tsdb-8edbcb8-marine-ad06.a.timescaledb.io
Database [postgres]: defaultdb
Port [5432]: 25145
Username [postgres]: tsdbadmin
Passwort für Benutzer tsdbadmin: rdbt9yr168bi80jw
```

This only worked after setting the pgAdmin SSL properties and loading the database. The certificate seems to be saved in the system, perhaps encrypted, since I could not find it anywhere.

On Linux, with psql

Put the ca.pem certificate in `~/geomar/certificates`. Then in bash, go to `~/geomar/certificates` and enter:

```
PGSSLMODE=require PGSSLROOTCERT=ca.pem psql -h tsdb-8edbcb8-marine-ad06.a.timescaledb.io -p 25145 -U
 tsdbadmin -d marinedb
```

Once the certificate has been loaded in the environment variables like this, it seems to hold for roughly half an hour before you have to load it again into the `PGSSLROOTCERT`, but that could also have come because I might have overwritten the var with a wrong value in the meantime.

```
PGSSLMODE=require psql -h tsdb-8edbcb8-marine-ad06.a.timescaledb.io -p 25145 -U tsdbadmin -d defaultd
b
```

API for the mobile frontend to the PostgreSQL Webserver backend

This is mainly the API with some very small SQL parametrisation in the insert and delete command. A parametrization prototype is given also for the select command, but I did not yet get this to work (`ReferenceError: request is not defined`). It should still run later in the app with `npx create-react-app react-postgres` and perhaps, even the Premises that were used in the original code need to be there.

The main aim was the API to connect to the server at all and get the needed endpoint (JSON) format as a result in the browser.

The certificate that you need to save at /etc/certificates/ca.pem:

- ----BEGIN CERTIFICATE-----
  MIIEQTCCAqmgAwIBAgIUPVry8oOWmxx4eHNWRAlKvdPPMSkwDQYJKoZIhvcNAQEM
  BQAwOjE4MDYGA1UEAwwvZDk2ZDM4MzQtNmFmMS00NmFlLWEyMjUtZGMwMDE1MDU0
  YjJlIFByb2plY3QgQ0EwHhcNMjEwOTA2MDQyMDA3WhcNMzEwOTA0MDQyMDA3WjA6
  MTgwNgYDVQQDDC9kOTZkMzgzNC02YWYxLTQ2YWUtYTIyNS1kYzAwMTUwNTRiMmUg
  UHJvamVjdCBDQTCCAaIwDQYJKoZIhvcNAQEBBQADggGPADCCAYoCggGBALUjayZk
  aZ5yjj3vjbtyPe8okGHniMwpII/gyTuhuQcqu2rxcZPDRlkgAjgYYVwmx+pf8DVY
  A2xNB8Lxol+Kt2kBHW24/dBzD3UOqmDcwSgufwYtSB1Ql9vaWWtCPgod1uf31mSv
  BW+XyoUYY/KRrK9+GslQTn70xDL5vxFumqLe9MkAS28NdPk2Gl7aNHKrEHAMCsl9
  MOEgbe6tuKFbfYjMUrkGNaYXSpEok92gvKRcYhoK+ocgS8aQEuSS6ah2bqnF8Cf0
  rYyXIC3CUdkVL1/mDTTVgaJb+2k2HisIZ86/oauuYepE5zhMaVp8zrfTz2LcB9Wv
  878EYO0zwiggxjI7AHMiqooZaxPlhi7vvTWaqGJkLhK0NY0sthlOqEX14MFQJKO4
  9m4WH1zcntGIxOTPV3vzsOIq86SWSctkd3HipM2CXgbbSaGbRQO5SHTqomhJu7p+
  TYIsL1bW17ddG1BDLXxgPbOYpX+XuxM8Z6CPUzLxv9+aSQoVGTdfFFNGEwIDAQAB
  oz8wPTAdBgNVHQ4EFgQUqOut8Uppgng2BLIekQ8WQOtFPc8wDwYDVR0TBAgwBgEB
  /wIBADALBgNVHQ8EBAMCAQYwDQYJKoZIhvcNAQEMBQADggGBAJrnDwKGvzV3FKd7
  +OylOI+M/PI4MnYKDtyECTlOVmIIJzJ6IrbqhZ04gH/uGs4xGpAxmAtSwx6s9AdR
  U8wIkhmcb9ytWFC/hyoK7v1EDEJWidxhTRtooE01y/mdZYiTFKOL9L0f/8H0tlGf
  V937TBZjmp4jyINMN/urR2LSiNYYuJo2ZgiAZv8Lw8TfKbolgzqstRCUn2G1MYFg
  HKTehiiBMoOaixSZ3K1Kq3o7JBwAT4r1eo2rz4FDNMV37vP52K2BpA/h3259bBTE
  ig7lTNeb2bxUBR2gmDj6uy0UTmErfv9HBb0mtqEq3H4y1hG0vnfXH6jPf7U5LeEq
  MAgLOX3gFVvBjM3eTc6LZugniyNK99114CSU8vKW08A6NNRHFLIFK7+cRjv76wSo
  KLXEfYabqoagaw/2EhrC2UfpBfdpMYAf/FQCaRFB+lPRrm/ds6gHbh94g03UpUW1
  5oFysnUa2wmcqyqv86bVcPfQfupJ383rXxtRwOPcVOEdP4eSdg==
  -----END CERTIFICATE-----

Of course, you can also save it elsewhere, but then you also need to change the path in the Pool class of the code below.

Following the guide at Getting started with Postgres in your React app: an end to end example where you also find more about the code that was used here. For digging deeper into this, How to quickly build an API using Node.js & PostgreSQL might help as well.

The Webserver must be running to test this. You need to follow the guide at first and install node and express.

Then create two files, `merchant_model.js` and `index.js`

merchant_model.js:

```
const Pool = require('pg').Pool
const fs = require('fs');
const pool = new Pool({
  user: 'tsdbadmin',
  host: 'tsdb-8edbcb8-marine-ad06.a.timescaledb.io',
  database: 'marinedb',
  password: 'rdbt9yr168bi80jw',
  port: 25145,
  ssl: {
    ca: fs
      .readFileSync("/etc/certificates/ca.pem")
      .toString()
  }
});

const getStokes = () => {
  const obs = parseInt(request.params.obs)
  pool.query('SELECT * FROM stokes WHERE obs = $1 ORDER BY obs ASC', [obs])
}

// Comment this out when you only want the request on select query.
// But only this query which does not use a request can be shown in the test browser at localhost:300
1
// Therefore, this is still needed for testing.
const getStokes = () => pool.query('SELECT * FROM stokes ORDER BY obs ASC')

const createStoke = (body) => {
    const { obs, traj } = body
    pool.query('INSERT INTO stokes (obs, traj) VALUES ($1, $2) RETURNING *', [obs, traj])
}

const deleteStoke = () => {
    const obs = parseInt(request.params.obs)
    pool.query('DELETE FROM stokes WHERE obs = $1', [obs])
}

module.exports = {
  getStokes,
  createStoke,
  deleteStoke,
}
```

Side note: In contrast to the guide, I use an SSL connection so that I need to define `fs` at the start, see How to establish a secure connection (SSL) from a Node.js API to an AWS RDS.

Now, being in the project directory, you can run it with `node index.js`.

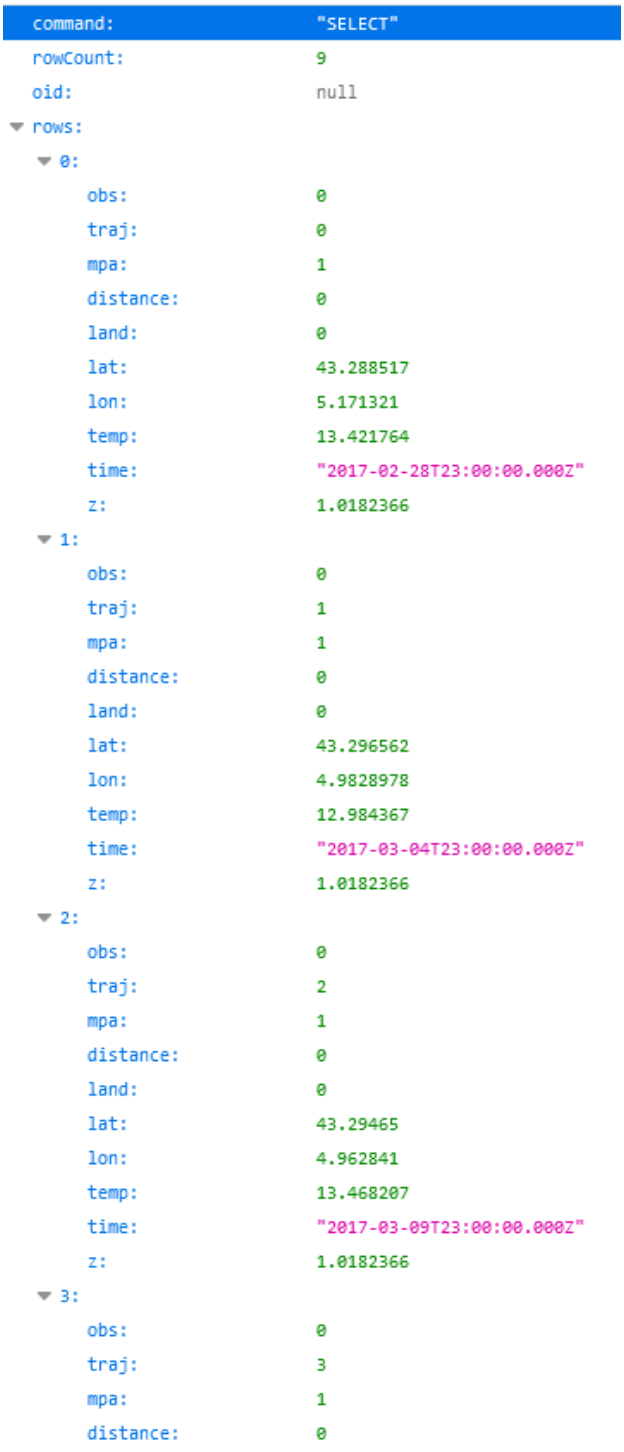
The `index.js`, a full copy of the guide, only getStokes gets run, it seems:

```
const express = require('express')
const app = express()
const port = 3001

const Stoke_model = require('./marine_model')

app.use(express.json())
app.use(function (req, res, next) {
  res.setHeader('Access-Control-Allow-Origin', '<http://localhost:3001>');
  res.setHeader('Access-Control-Allow-Methods', 'GET,POST,PUT,DELETE,OPTIONS');
  res.setHeader('Access-Control-Allow-Headers', 'Content-Type, Access-Control-Allow-Headers');
  next();
});

app.get('/', (req, res) => {
  Stoke_model.getStokes()
  .then(response => {
    res.status(200).send(response);
  })
  .catch(error => {
    res.status(500).send(error);
  })
})

app.post('/Stokes', (req, res) => {
  Stoke_model.createStoke(req.body)
  .then(response => {
    res.status(200).send(response);
  })
  .catch(error => {
    res.status(500).send(error);
  })
})

app.delete('/Stokes/:id', (req, res) => {
  Stoke_model.deleteStoke(req.params.id)
  .then(response => {
    res.status(200).send(response);
  })
  .catch(error => {
    res.status(500).send(error);
  })
})
app.listen(port, () => {
  console.log(`App running on port ${port}.`)
})
```

If that runs, you should see

```
App running on port 3001.
```

in the command prompt, with the command still running. Then you open a browser and enter `localhost:3001` to see



if you do not use a where condition on the select (the test table has just 9 rows).

App

The React app on top of this is yet to come. Perhaps, it will even need the `Premises` that were used in the original merchant_model.js.

It will be a GUI where you can enter the obs so that you can control what you want to insert or delete (or, if also parametrised, what to delete).

DONE

After following the installation guide (remove the content of react-postgres/src dir and put App.js + a new index.js in the react-postgres/src dir), I can send the parametrised requests.

In the project dir:

```
npx create-react-app react-postgres
```

App.js to be put in the subfolder /src:

```
import React, {useState, useEffect} from 'react';
function App() {
  const [stokes, setStokes] = useState(false);
  useEffect(() => {
    getStoke();
  }, []);


  function getStoke() {
    fetch('<http://localhost:3001>')
      .then(response => {
        return response.text();
      })
      .then(data => {
        setStokes(data);
      });
  }
  function getStoke() {
    let obs = prompt('Enter stoke obs');
    fetch('<http://localhost:3001>')
      .then(response => {
        return response.text();
      })
      .then(data => {
        setStokes(data);
      });
  }
  function getStoke() {
    let obs = prompt('Enter stoke obs');
    let traj = prompt('Enter stoke traj');
    fetch('<http://localhost:3001>', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({obs, traj}),
    })
      .then(response => {
        return response.text();
      })
      .then(data => {
        setStokes(data);
      });
  }




  function createStoke() {
    let obs = prompt('Enter stoke obs');
```

```
    let traj = prompt('Enter stoke traj');
    fetch('<http://localhost:3001/stokes>', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({obs, traj}),
    })
      .then(response => {
        return response.text();
      })
      .then(data => {
        alert(data);
        getStoke();
      });
  }
  function deleteStoke() {
    let obs = prompt('Enter stoke obs');
    fetch(`http://localhost:3001/stokes/${obs}`, {
      method: 'DELETE',
    })
      .then(response => {
        return response.text();
      })
      .then(data => {
        alert(data);
        getStoke();
      });
  }
  return (
    <div>
      {stokes ? stokes : 'There is no stoke data available'}
      <br />
      <button onClick={createStoke}>Add stoke</button>
      <br />
      <button onClick={deleteStoke}>Delete stoke</button>
    </div>
  );
}
export default App;
```

index.js to be put in the subfolder /src:

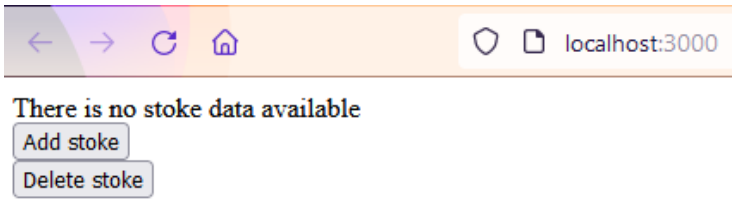
```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';

ReactDOM.render(<App />, document.getElementById('root'));
```

Then in the /react-postgres dir:

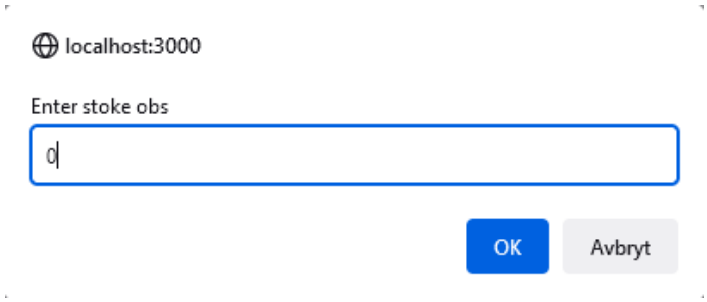
```
npm start
```

and you get:

and a browser opens:



asking you to:



Entering 0 here to select all obs with value 0 does not show any data, but it normally should show the full table of 9 rows since all obs are 0. (TODO)

This is still almost finished.
The create does not work since the full table columns would need to be inserted, not just obs and traj (I guess so).

It should be easy to find out about this since there is a Merchant table as the example that would probably work. And then it is a small step to add other columns to the parameters.