

Max-Objects Database

Suverino Frith and Oliver Rayner

Backend (server side)

The backend is made up of three parts: PostgreSQL, GraphQL, and NodeJS. The database is hosted on a PostgreSQL server. There is a tool named PostGraphile that takes a PostgreSQL database's schema and automatically generates an immediately available GraphQL API. PostGraphile then acts as a middleware to connect NodeJS to the GraphQL API which can then be used by the frontend application.

The first step to making this all work was migrating the database code to a syntax acceptable by PostgreSQL. The old database code was written to work on MySQL. In order to accomplish this we created an Ubuntu 18.04 virtual machine that we knew could host all of the softwares required for the migration: MySQL version 5.6, pgloader version ≤ 3 , and PostgreSQL ≤ 10 . pgloader is a program that actually takes an existing MySQL database and migrates it to an existing PostgreSQL database. The reason we are using such an old version of MySQL is that pgloader is not compatible with the new way MySQL authenticates its users, and this is the newest version without that new authentication program.

On Ubuntu 18.04 it was necessary to download MySQL from the source tree using community edition of MySQL available on their github, and then to build and install it from source. PostgreSQL and pgloader were available from the default Ubuntu repositories. Once all of these three softwares were acquired, we created a database named 'max' in both MySQL and PostgreSQL. We then ran the database script that was provided to our group on the MySQL database after modifying one line in the script that was giving pgloader problems. And finally we ran pgloader while supplying it with both databases as can be seen in figure 1. This provided us with a PostgreSQL database that was a replica of the original MySQL database. We then made PostgreSQL generate a script that could recreate the database with PostgreSQL compatible SQL code. Following this we moved the script to our development repository for use on our development machines.

```

seafang@neptune-red18:~$ pgloader mysql://root@localhost/max pgsq://max
2021-05-02T00:15:49.108000Z LOG report summary reset

```

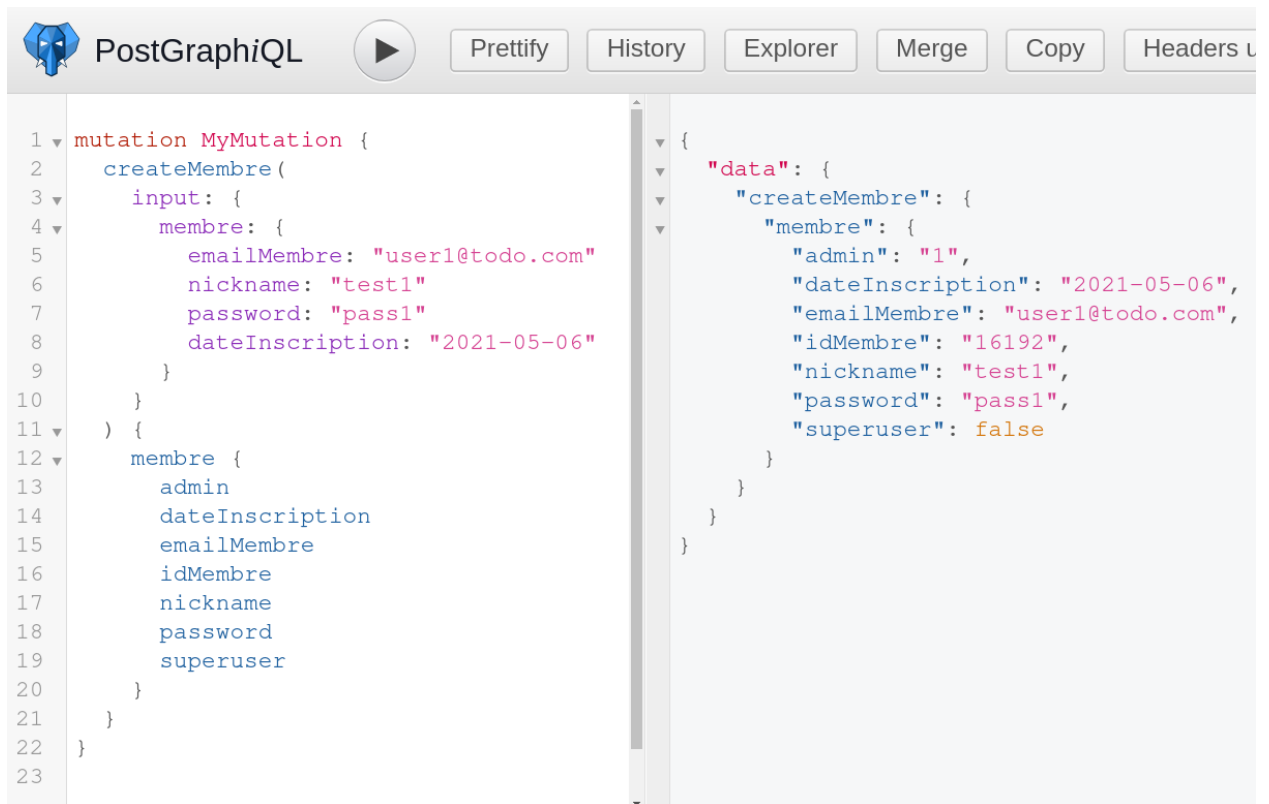
table name	read	imported	errors	total time
fetch meta data	105	105	0	0.085s
Create Schemas	0	0	0	0.001s
Create SQL Types	0	0	0	0.006s
Create tables	52	52	0	0.074s
Set Table OIDs	26	26	0	0.005s
max.auteurs	343	343	0	0.024s
max.auteur_libraries	161	161	0	0.016s
max.commentaires	57	57	0	0.018s
max.formats	11	11	0	0.025s
max.library_plateformes	288	288	0	0.039s
max.library_wares	142	142	0	0.053s
max.membres	16178	16178	0	0.265s
max.auteur_objets	5995	5995	0	0.067s
max.environnements	7	7	0	0.033s
max.libraries	135	135	0	0.056s
max.library_url	154	154	0	0.059s
max.maxversions	8	8	0	0.092s
max.news	110	110	0	0.106s
max.objet_environnements	8165	8165	0	0.135s
max.objet_libraries	4151	4151	0	0.122s
max.objet_plateformes	10604	10604	0	0.134s
max.objet_wares	5116	5116	0	0.111s
max.objets	4851	4851	0	0.159s
max.sessions	149	149	0	0.139s
max.url	446	446	0	0.157s
max.objet_formats	5126	5126	0	0.087s
max.objet_maxversions	7310	7310	0	0.096s
max.objet_url	5310	5310	0	0.067s
max.plateformes	4	4	0	0.056s
max.sondage	4	4	0	0.054s
max.wares	4	4	0	0.066s
COPY Threads Completion	4	4	0	0.384s
Create Indexes	79	79	0	1.137s
Index Build Completion	79	79	0	0.071s
Reset Sequences	13	13	0	0.017s
Primary Keys	3	3	0	0.002s
Create Foreign Keys	0	0	0	0.000s
Create Triggers	0	0	0	0.000s
Install Comments	0	0	0	0.000s
Total import time	74829	74829	0	1.114s

Figure 1: pgloader migrating the database

The next step was to automatically generate a GraphQL API to interface with the database. To do this we used PostGraphile because it can instantly create a GraphQL API when pointed at an existing PostgreSQL database. We imported postgraphile into our NodeJS application, gave it the information it needed to find our locally hosted PostgreSQL database,

assigned it a port to run on, and we were good to go. We now had a fully functional GraphQL API that could serve our frontend application using NodeJS and ExpressJS.

For most of our purposes the base API generated was enough. In order to register new members we used the 'createMembre' mutation provided by PostGraphile. We used GraphiQL to test our API as can be seen in figure 2. GraphiQL is a user interface designed to be a quick and easy way to test GraphQL APIs. Then we used pgAdmin (a user-interface for PostgreSQL) to test that the API successfully updated the database. We viewed the database by giving pgAdmin specific queries as seen in figure 3. With both tests being successful, we called the API from the frontend after enabling cross origin request support (CORS), and this can be seen in figure 4. CORS is necessary to make two ports on the same machine communicate with each other (the port running the backend [:8080] and the port running the frontend [:3000]).



```
PostGraphiQL [Play] [Prettify] [History] [Explorer] [Merge] [Copy] [Headers U]

1 mutation MyMutation {
2   createMembre(
3     input: {
4       membre: {
5         emailMembre: "user1@todo.com"
6         nickname: "test1"
7         password: "pass1"
8         dateInscription: "2021-05-06"
9       }
10    }
11  ) {
12    membre {
13      admin
14      dateInscription
15      emailMembre
16      idMembre
17      nickname
18      password
19      superuser
20    }
21  }
22 }
23

{
  "data": {
    "createMembre": {
      "membre": {
        "admin": "1",
        "dateInscription": "2021-05-06",
        "emailMembre": "user1@todo.com",
        "idMembre": "16192",
        "nickname": "test1",
        "password": "pass1",
        "superuser": false
      }
    }
  }
}
```

Figure 2: Testing the API with GraphiQL

	id_membre bigint	superuser boolean	nickname character varying (50)	password character varying (25)	email_membre character varying (70)	date_inscription date	admin bigint
1	16192	false	test1	pass1	user1@todo.com	2021-05-06	1
2	16191	false	test3	pass	test3@todo.com	2021-05-04	1
3	16190	false	suv	pass	suv@todo.com	2021-05-04	1
4	16189	false	test2	password	max2@todo.com	2021-05-02	1
5	16188	false	test1	pass	max@todo.com	2021-01-02	1

Query Editor Query History

1 select * from max.membres order by id_membre desc limit 5

Figure 3: Viewing the database via queries on pgAdmin

```
const requestBody = {
  query: `
    mutation {
      createMembre (
        input: {membre: {emailMembre: "${email}", password:
"${password}", nickname: "${username}", dateInscription:
"${formattedDate}"}}
      ) {
        membre {
          admin
          dateInscription
          emailMembre
          idMembre
          nickname
          password
          superuser
        }
      }
    }
  `
};

fetch("http://localhost:8080/graphql", {
  method: 'POST',
  body: JSON.stringify(requestBody),
  headers: {
    'Content-Type': 'application/json'
  }
})
```

Figure 4: Calling the API in ReactJS

We created a custom SQL function to support login functionality. This was advantageous because it allowed us to minimize the amount of data sent to and received from the database when authenticating a member's login. Our custom function took in a user's password, and either their email or username, and returned the userID only if the information provided matched a row in the database; otherwise it returned null. The function can be seen in figure 5. PostGraphile automatically took our newly created SQL function and added it to our GraphQL

API, and so we were able to immediately test it using GraphQL and pgAdmin, and implement the function into our application. After the users were able to login we used the React-Redux library to store the user's ID throughout the application so that any React component could call the GraphQL API to send and receive data related to the current user. This concludes the makeup of our backend NodeJS server.

```
Dashboard Properties SQL Statistics Dependencies Dependents max/postgres...
1 -- FUNCTION: max.login(text, text, text)
2
3 -- DROP FUNCTION max.login(text, text, text);
4
5 CREATE OR REPLACE FUNCTION max.login(
6     username text,
7     email text,
8     pass text)
9     RETURNS bigint
10    LANGUAGE 'plpgsql'
11    COST 100
12    VOLATILE PARALLEL UNSAFE
13 AS $BODY$
14 declare iden bigint;
15 begin
16     select id_membre into iden
17     from max.membres
18     where (nickname=username or email_membre=email) and password=pass;
19     return iden;
20 end;
21 $BODY$;
22
23 ALTER FUNCTION max.login(text, text, text)
24     OWNER TO postgres;
25
```

Figure 5: Custom SQL login function

Frontend (client side)

The frontend is made up from CSS & JSX files using the React framework. The website is set up in React and uses a quick web development tool called react-bootstrap which is already part of the website (<https://react-bootstrap.github.io>). Bootstrap is pretty easy to learn and is a great tool for setting up good looking components quickly with bootstrap by using certain commands available on the website -- link above -- and using className with CSS styling.

To start off you will need to have knowledge of HTML, CSS, and JavaScript (JS). HTML and CSS are pretty quick to understand, and for JS you will just need to know the basics in order to progress. Look over the React tutorials next and see how JSX takes both HTML and JS to make websites and how the className defines certain attributes. All of the recommended videos for getting started are listed on the frontend/README.

You will also learn by reviewing the current project, try to find the patterns by looking at the website and how the code changes the UI. Also the keywords on the bottom of the page will help you understand the concepts that you need to get started with the frontend's future development. Along with the prototyped images provided on the Wiki page Max Objects site, you will know the direction on how to configure the website. Figure 6 shows our most complex page and it was used with a combination of react-hooks, react-bootstrap, and a variety of components. With a little bit of time and reviewing the MyObjects.jsx file you should be able to understand what is happening.

Figure 7 shows the registration page which is what we first used to connect the frontend with the backend. Many input fields were created and looking over the register and login jsx files you will find that they are pretty similar in the way they handle the different form components. This was also created primarily using react-bootstrap. The current look is very basic but can be adjusted using CSS which can be followed along to on the frontend/README.

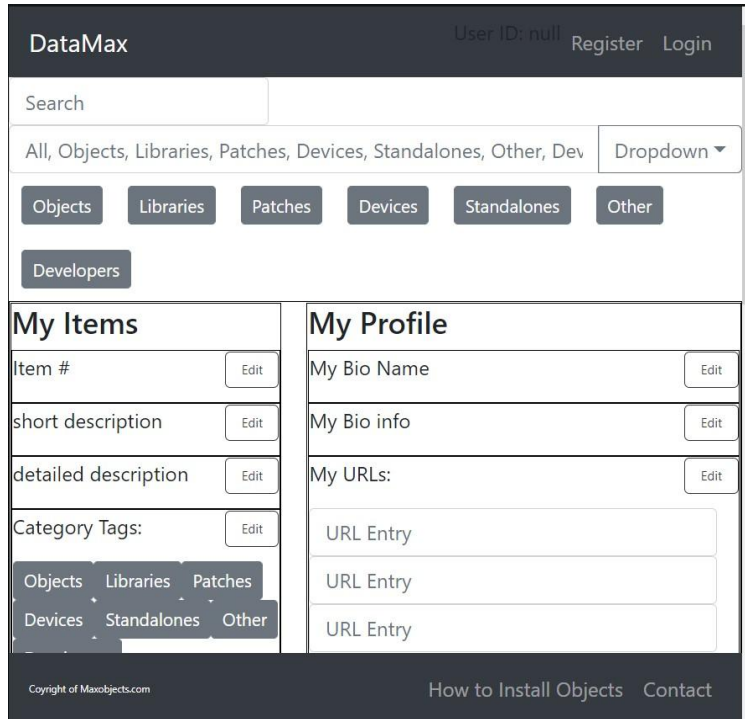


Figure 6: localhost:3000/myobjects

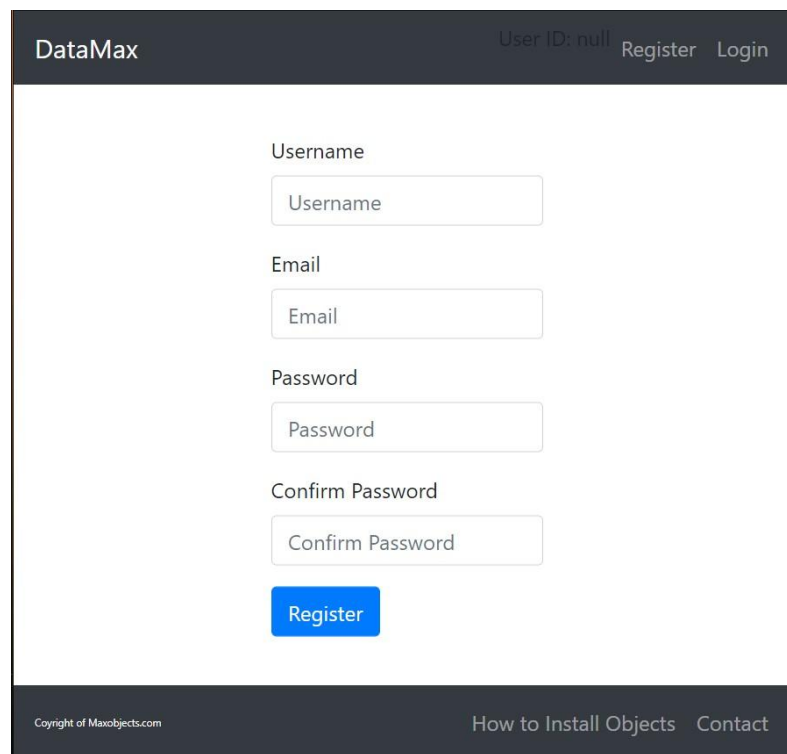


Figure 7: Registration page

When you want to add additional pages that will show up on the website. You will need to put that link within the router. The current website uses react-router (which is what we are using for the website) which can be found in the App.jsx file.. Here as shown in figure 8, the "/" followed by the name creates a new page that can be accessed on the localhost:3000. For example typing localhost:3000/myobjects in the search bar will bring you to the myobjects page which runs on the MyObject.jsx file.

Sometimes you will need to import files from different sections and the basic example in figure 9 shows what that looks like on the top of the App.jsx file. You will notice when starting to code that there are many instances of importing and it is the only way to connect to files within the text editor. If it is confusing I would recommend looking up about react-router on youtube.

```
<Switch>
  <Route path="/" exact component={Home}/>
  <Route path="/register" exact component={Register} />
  <Route path="/login" exact component={Login}/>
  <Route path='/myobjects' exact component={MyObjects}/>
  <Route path="/hdiio" exact component={HDIIIO}/>
  <Route path="/contact" exact component={Contact}/>
  <Route path='/objects' exact component={Objects}/>
  <Route path='/state' exact component={State}/>
  <Route path='/props' exact component={Props}/>
</Switch>
```

Figure 8: React-Router section on App.jsx

```
import React, { Component } from 'react';
import './App.css';
import 'bootstrap/dist/css/bootstrap.min.css'
import Contact from "./Components/Contact"
import HDIIIO from "./Components/HDIIIO"
import Home from "./Components/Home"
import Navigation from "./Components/Navigation";
```



```

import Register from "./Components/Register";
import Login from "./Components/Login";
import Footer from "./Components/Footer"
import Objects from "./Components/Objects"
import MyObjects from "./Components/MyObjects";
import State from "./Components/StateEX";
import Props from "./Components/PropsEX";
import {BrowserRouter as Router, Switch, Route } from "react-router-dom";

```

Figure 9: importing example on App.jsx

Props are used in order to get the components to change based on user inputs on a component. In the project I put an example of this that can be run by doing the following, starting up React with yarn start (mentioned in the backend portion) and then entering localhost:3000/props into the search bar. By doing that figure 12 should appear on the web browser. You will notice that the text is not editable and is stationary. Looking at figure 10 and figure 11, you should be able to spot out some patterns. Figure 10 shows how the component <Superhero/> is made and figure 11 shows the manipulation of the component by using different name and age attributes. Props are very useful for making your own components and if you mess around with both pages, adding new prop attributes and adding more <Superhero/> components, you will have gotten a good grasp on the concept of props.

```

import React from 'react';

//Used as example for showing prop functionality. You can use this for coding as
a template

//or you can delete all files with EX behind them and remove them out of the
App.jsx file if you do not need them.

const Hero = props => {
  return <h1> This is {props.hero} and they are {props.age}</h1>
}

export default Hero;

```

Figure 10: SuperHeroPropEX.jsx

```

import React from 'react';

```

```
import './Home.css';
import 'bootstrap/dist/css/bootstrap.min.css';
import Superhero from './SubComponents/SuperHeroPropEX';

//Used as example (use /props at end of localhost:3000 to access) for showing how
props work. You can use this for coding as a template

//or you can delete all files with EX behind them and remove them out of the
App.jsx file if you do not need them.

function PropEX() {
  return (
    <div>
      <Superhero hero = "Batman" age = "34"/>
      <Superhero hero = "Superman" age = "35"/>
    </div>
  );
}

<div className='news'>
</div>

export default PropEX;
```

Figure 11: PropsEX.jsx

This is Batman and they are 34
This is Superman and they are
35

Figure 12: localhost:3000/props on web browser

States are used to change the layout after certain user actions. In the StateEX.jsx file there are also examples of react-bootstrap. To get to the state example page, open up the localhost:3000 website with yarn start and type localhost:3000/state in the search bar. Figure 14 should show up on the web browser at this point. You will notice that useState and the {Button} for react-bootstrap are imported in and this will be how you will get the functionality. Look at `const [count,setCounter] = useState(2);` in figure 13 and you will notice that the same number in useState(2) is shown on the website. Each button has a functionality and is connected to a different bootstrap variant color. The increment button increases the number, the decrement button decreases the number, the nothing button does not have functionality, and the zero button sets the code back to zero. The setCount changes the count and count is what the number currently is.

```
import React, {useState} from 'react';  
  
import "./Home.css";
```

```
import 'bootstrap/dist/css/bootstrap.min.css';

import {Button} from 'react-bootstrap';

//Used as example (use /state at end of localhost:3000 to access) for showing
react hooks and state. You can use this for coding as a template

//or you can delete all files with EX behind them and remove them out of the
App.jsx file if you do not need them.

function StateEX() {

  const [count,setCounter] = useState(2);

  function add() {

    setCounter(count + 1)

  }

  function minus() {

    setCounter(count - 1)

  }

  function zero() {

    setCounter(0)

  }

  return (

    <div>

      <Button variant = "success" onClick={add}>Increment</Button>

      <p>{count}</p>

      <Button variant = "danger" onClick={minus}>Decrement</Button>

      <Button>Nothing</Button>

    </div>

  )
}
```

```
    <Button variant = "dark" onClick={zero}>Zero</Button>

  </div>

);

}

<div className='news'>

</div>

export default StateEX;
```

Figure 13: StateEX.jsx

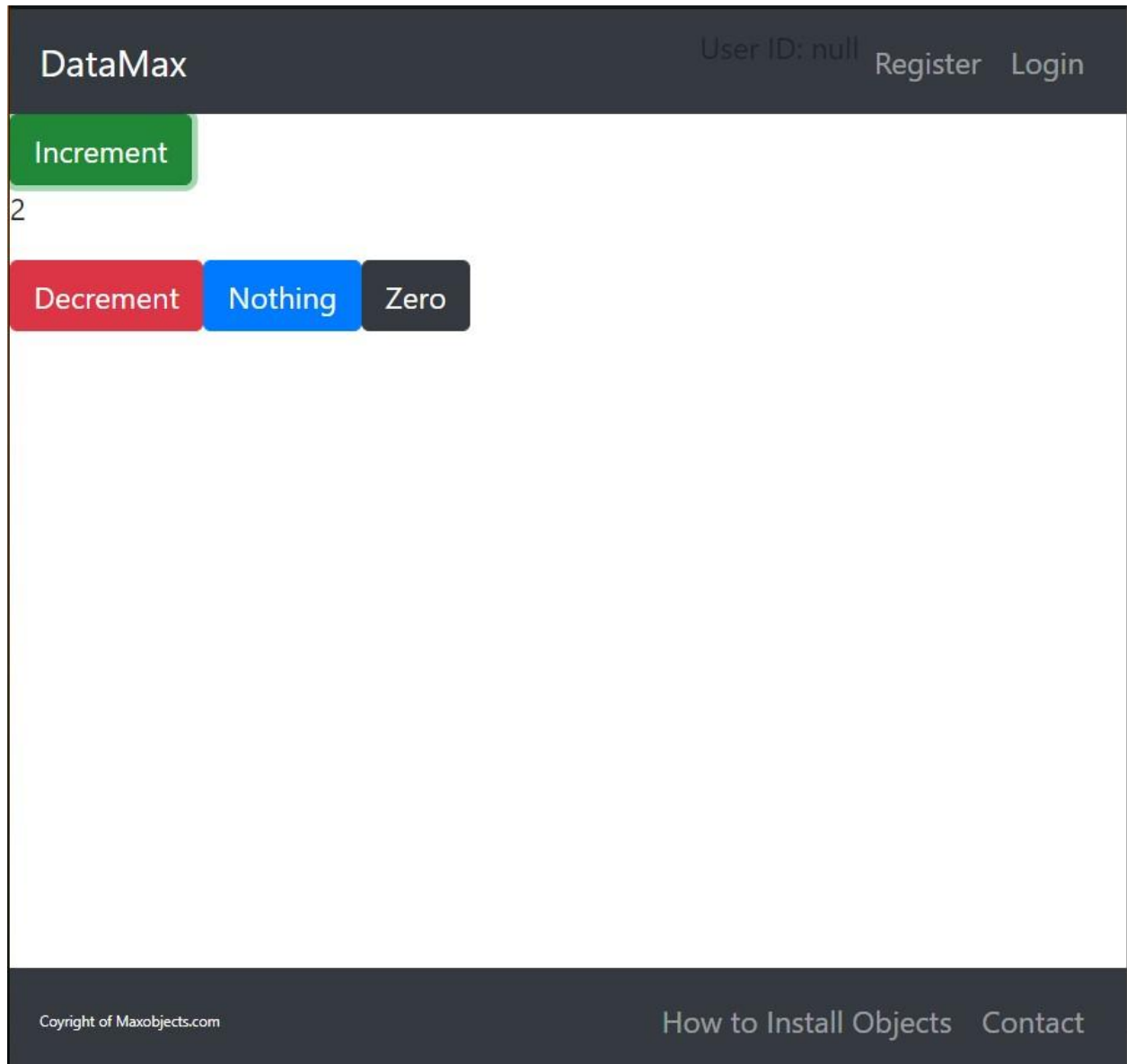


Figure 14: localhost:3000/state on web browser

If you have any confusions while coding ahead make sure to search for the answer or find a friend who has experience with coding. The examples are pretty basic but should give you a strong foundation going forward. React takes time to learn but is a very useful tool. I highly recommend looking through the backend and frontend READMEs in order to have a head start on development.

Next Steps

Resources

The resources necessary to learn the appropriate skills required for continued development of this project are available in the respective backend and frontend folders of the revised project repository.

Known Issues

The global state of the application resets due to automatic page refreshes whenever you click on a react component. This is an issue because whenever we store a user ID that can be used throughout the entire app, it is immediately erased when the user switches pages to a page where the ID would be useful. There is a way to configure the frontend server so that this does not happen, but we were unable to configure it in such a way using the time we had.

Backlog

We used the Jira software to keep track of what updates we need to make to the application, the amount of effort those updates will take (the number to the far right), and the priority level of each task (the colored arrows on the far right). This can be seen below.

Backlog 18 issues Create sprint ...

<input type="checkbox"/> Allow users to add objects	MOW-2	↑	6
<input type="checkbox"/> Allow users to create libraries	MOW-3	↑	8
<input type="checkbox"/> Allow users to do a general search	MOW-13	↑	8
<input type="checkbox"/> Allow users to view other users projects	MOW-4	↑	6
<input type="checkbox"/> Allow users to edit their items	MOW-22	↑	7
<input type="checkbox"/> Allow users to edit their profile	MOW-23	↑	3
<input type="checkbox"/> Allow users to give an item multiple tags	MOW-9	↑	5
<input type="checkbox"/> Allows users to search by category	MOW-14	↑	5
<input type="checkbox"/> Have video uploading options	MOW-15	↓	7
<input type="checkbox"/> Have images uploading options	MOW-24	↓	4
<input type="checkbox"/> Allow maintainers to designate Random {Blank} of the Day	MOW-21	↓	3
<input type="checkbox"/> Allow maintainers to edit news	MOW-20	↓	4
<input type="checkbox"/> Allow superusers to moderate posts	MOW-11	↓	5
<input type="checkbox"/> Deploy on DigitalOcean	MOW-31	↓	5
<input type="checkbox"/> Allow users to visit "How Do I Install Objects" page	MOW-18	↓	1
<input type="checkbox"/> Make UI styled like cycling 74 website	MOW-12	↓	3
<input type="checkbox"/> Make website navigable on mobile	MOW-5	↓	4
<input type="checkbox"/> Create system for users to find ways to get started with unfamiliar max objects			

Quickstart

Key Words:

React: Framework for the website.

HTML (HyperText Markup Language): Uses tags with <> to place basic elements with attributes with = signs. For example `<p id="red">` This is a paragraph `</p>`, yellow is tag and blue is attribute.

CSS (Cascading Style Sheets): Uses selectors to change visual aspects of the HTML. For example `.red { color: red}`, yellow selects the item to change and blue modifies it to red.

JS (JavaScript): Basic programming language to give functionality to the website.

JSX: Used in React and combines HTML and JS into one.

React-Bootstrap: Used to make fast and good-looking components quickly.

React-Router: Organizes the set up between the different pages.

Props: Used for changing component variables but must be used on a different file.

State: Must be used on the same file and updates and changes the form based on different inputs.

PostgreSQL: An open source relational database

GraphQL: An API framework that simplifies requests to the backend and provides a flexible and standardized data format

PostGraphile: A middleware that takes a PostgreSQL database and generates a GraphQL API for it

NodeJS: A javascript runtime environment created for running javascript outside of the browser

ExpressJS: A minimalist web framework for node

React-Redux: A predictable state container for javascript applications