Programming in Geomatics, TBA4251
2016

Mathilde Ørstavik

CONTENTS

1

INTRODUCTION

Spatial analysis are more and more used in many different contexts and applications. For many years GIS-applications have been mainly for desktop, but the latest years there have been developed more libraries for supporting spatial analysis in web applications as well.

Web applications are in many ways ”the future” of applications, and I therefor chose to create a web application for my project in the course ”Programming in Geomatics”. I wanted to see what kind of a GIS I would be able to create using all the fantastic open source libraries out there.

This report will cover the process of developing the web application ”GISTYLE” for the course ”Programming In Geomatics TBA4251”. ”GISTYLE” is a wordplay for the three words; GIS, style and tile.

2

GOALS OF THE APPLICATION

The overall goal of the application GISTYLE consist of three main parts.

The first goal was that the application should be easy to use, which means that a good user interface and user experience was a priority.

As well as being easy to use I also wanted it to be easy to create good visualisations from the results of the analysis, so styling the data should be easy. It doesn't matter if the analysis are good, if the map isn't visualising the results in an understandable way.

The third important goal for my application was to save progress from projects within the application, meaning the application needed log-in functionality, user authentication and to save all the user activity to a database.

3

PROGRAMMING LANGUAGES, DATABASE AND SERVER

To be able to reach the goals of my application, it was important to make good choices for what tools and programming languages to use. There were some limitations due to the fact that all tools needed to be free, especially when it came to database and server. I will now go through the tools and programming languages I ended up using, and explain some of the reasons they were good choices.

3.1 HTML AND SCSS

When creating a web application CSS and HTML are needed to create the views. HTML defines the elements of the view, and CSS defines the styling. Instead of writing regular CSS, I used SCSS which gives some extra functionality, like variables and nested selectors.

```
$font-stack:    Helvetica, sans-serif;
$primary-color: #333;

body {
  font: 100% $font-stack;
  color: $primary-color;
}
```

Figure 1: Example of SCSS code showing the use of variables

3.2 THE MEAN STACK

The mean stack [2] is a full stack JavaScript solution that helps build fast, robust and maintainable web applications. MEAN stands for MongoDB, Express, Angular and Node. It unifies the language and data format (JSON), which gives many advantageous. The data flows neatly between all the layers of the application without having to rewrite or reformat. There is a lot of talk about the mean stack, and it has become quite popular the last two years.

3.2.1 *MongoDB*

MongoDB [3] is a free and open-source cross-platform document-oriented database. It is a noSQL database, which means that it avoids the traditional table-based relational structure, and instead uses a JSON-like structure called BSON. It has elastic scalability, high performance and it is a flexible database.

3.2.2 *Express*

Express [7] is minimal and flexible web framework for Node. It is responsible for providing the Web API that empowers the MEAN stack.

3.2.3 *Angular.js*

Angular [5] is a JavaScript framework, which makes it easy to create dynamic single-page-application. You can say that it lets you extend the HTML vocabulary for your application.

3.2.4 *Node.js*

Node[6] is an open-source, cross-platform runtime environment for developing server-side Web applications. Node has an event-driven architecture capable of asynchronous I/O, which means it can build fast and scalable network applications.

3.3 HEROKU

For deploying the application the cloud application platform, Heroku [1], was used. It offers a free ”Hobby” user, that gives you 25 MB RAM and up to 20 connections. Heroku lets you deploy, run and manage applications written in different programming languages, and it was therefor a good solution for my project.

4

TOOLS AND LIBRARIES

A good rule of thumb for any developer is to be as lazy as possible. If you don't have to write the code yourself, don't do it! Using good open source libraries is therefor important, and crucial for being able to create an application like GISTYLE.

The libraries used in this project are many, but I will only explain the most important ones.

4.1 LEAFLET

Leaflet [8] is an open source JavaScript library and it is designed with simplicity, performance and usability in mind. It works efficiently across all major desktop and mobile platforms, it can be extended with lots of plugins and it has an easy to use and well-documented API.

In GISTYLE, Leaflet is used for drawing the map and the layers, and handles the interactivity with the map.

4.2 TURF.JS

Turf [10] is an open source JavaScript library for advanced geospatial analysis. The library is created by Mapbox, and it is based on JSTS, the JavaScript conversion of the Java library JTS. Turf has implementations of the most common geospatial operations, but it is still under development and is not complete yet.

4.3 GEOJSON-VALIDATION

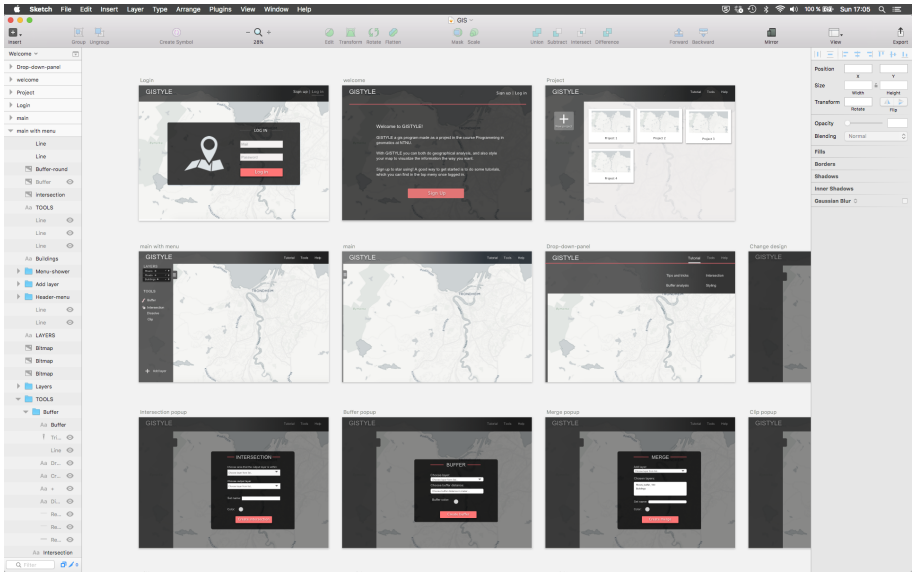
To make sure that the GeoJSON uploaded by the user is actually a valid GeoJSON file, the library GeoJSON-validation [14] was used.

4.4 PASSPORT

To add security to the application, the library Passport [18] was used. It is an authentication middleware for Node, and it takes care of the encryption of user passwords.

4.5 SKETCH

Another important tool used throughout the project was an application called Sketch [19]. It is a design tool, and it focuses entirely on user interface design. This tool was an important part in the workflow of creating the user interface for the application.



4.6 TRELLO

Trello [20] is an online tool that offers a kanban-style of working. You can define boards yourself, and easily switch the so called ”cards” between the different boards. Trello was used for planning and organising.

5

WORKFLOW AND ASSESSMENT OF IT

5.1 PLANNING

When I started working on the web application GISTYLE, I had very limited experience when it came to frameworks and back end programming. I therefor needed to read a lot before I could get started. Tutorials, articles and reading pros and cons about different tools was a key part of the first few weeks of work.

After deciding what programming languages and framework to use, I defined the functional requirements for the application. I decided on what geospatial analysis I wanted, what kind of design changes and interactivity with the map the user should be able to do and so on.

5.2 IMPLEMENTATION

5.2.1 *Backend*

I started implementing the back end of the application first. I deployed the app to Heroku, connected to the database and implemented the log in functionality with authentication of users and encryption of passwords.

5.2.2 *Frontend*

I then continued with the front end of the application. In general, this was done by first creating a prototype of the design using sketch (see 4.5). Testing the design in a design tool versus testing in HTML and CSS makes the work a lot easier and, in the long term, more efficient.

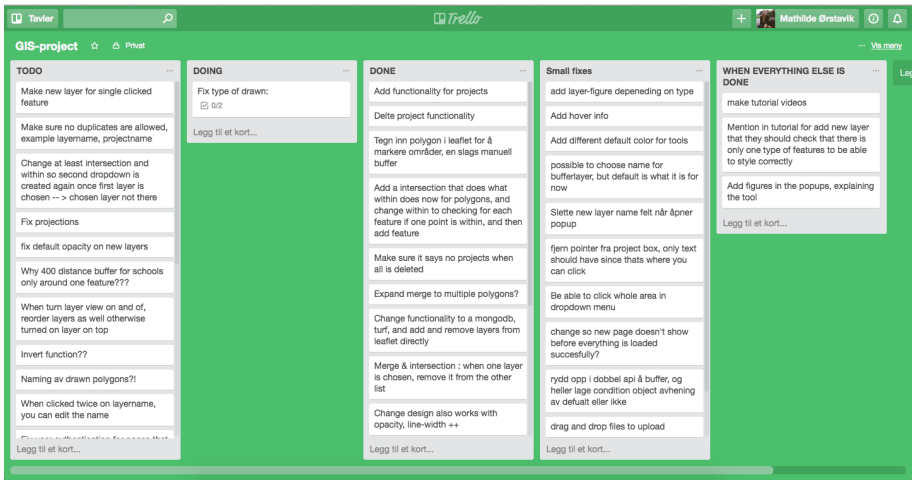
5.2.3 *Geospatial analysis*

After implementing some of the views, I started on the geospatial analysis. This turned out to be more complicated than first expected, and will be further discussed in chapter 6.

5.2.4 *Assesment*

Generally the work was well distributed throughout the semester. To make sure I used my time efficiently, and to keep track of what had to be done, I used "Trello" (see 4.6). I had five boards; "TODO", "DOING", "DONE", "Small fixes" and "WHEN EVERYHING ELSE IS DONE". This way I could easily order my todo's

according to what was most important. I also wrote down things I realised should be implemented and bugs I found to make sure I wouldn't forget them.



6

CHALLENGES DURING IMPLEMENTATION

The main challenge during implementation was how to implement the spatial analysis.

6.1 POSTGIS

When working with geospatial analysis, PostGIS is natural to consider using. PostGIS is an open source software program that adds support for geographic objects to the PostgreSQL [15]. It has support for spatial queries, and is an essential tool in geomatics. Unfortunately using PostgreSQL together with Heroku is only free for very small amounts of data.

One solution could have been to save all data in MongoDB, and for every spatial query I would need to run I could send the data for the specific query to the PostgreSQL database, run the query and save the results back to MongoDB. This way I would only need to store small amounts of data at once in PostgreSQL, but the data would be passed back and forth a lot which is not optimal and would slow things down.

Another solution that I tried to implement, was using PostGIS through CartoDB [16] for running the geospatial analysis. I added some default data layers to my personal CartoDB user, and ran the query from my web application. The problem with this was that I needed to save the resulting data from queries directly to my CartoDB user. This would have caused my user to be filled up with data, and the free user has a limit for how much data you can store which would have been exceeded fast. I did try to, instead of storing the resulting data, store the queries. This soon got very complicated, since it resulted in ”nested queries” as soon as the result from the one query was used in another analysis.

Security-wise it would also be a bad idea to give access to my CartoDB user.

6.2 JAVASCRIPT LIBRARIES FOR SPATIAL ANALYSIS: TURF.JS VS JSTS

After having attempted different approaches with PostgreSQL, I figured out that it would be better to use a JavaScript library for the spatial analysis in my project. The two candidates for this was JSTS [9] and Turf [10] (see chapter 4.2). I chose Turf since it is a very easy library to use, the learning curve isn't as steep as for JSTS, the documentation is better and the both use more or less the same algorithms [13]. Turf is GeoJSON from the ground up, and focuses on data analysis. Since it is still under development, it has been really interesting to follow the updates and see how it has improved during the time I have worked with the project.

One of the most frustrating problems during implementation was that turf had not solved the problem with projection error on buffers. All buffers was distorted, unless they were created around equator (see figure 2).

I tried to solve the problem by writing a script for converting the coordinates before creating the buffer, and for converting them back after the buffer was created.

Almost all the digital maps in web-applications use the web-mercator projection, also called EPSG:3857 [21]. It is very similar to the UTM-projection, but it is optimized for web maps. I therefor tried to convert the coordinates to EPSG:4326, which gives unprojected coordinates, then create the buffer, and at last project the coordinates for the buffer back to the web-mercator projection before visualising the result. I also tried to convert the coordinates to the UTM-projection in zone 32, since the coordinates I tested are all in that zone. The code for the projection script can be found at GISTYLE/projectionTest.

None of these two approaches worked, and the result was exactly the same as if I didn't convert the coordinates at all.

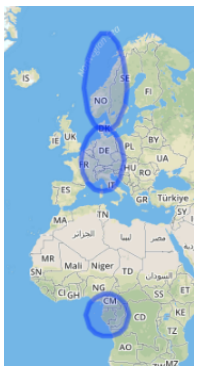
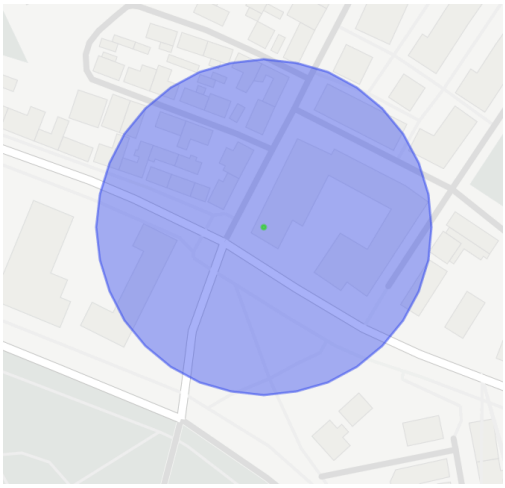


Figure 2: Visualisation of how the buffer is distorted. The further away from equator, the bigger the distortion.

After doing a lot of research, and reading through the turf-buffer code [11], I discovered that I was far from the only one frustrated with the projection errors in the turf buffer. I even found someone who had taken the problem into his one hands, and started to implement a fully geodesic buffer [12]. I checked up on his progress regularly, and after some months he had managed to write a script that was working. The only thing missing is more testing, and then turf will probably add it to their library permanently. I started testing the code myself and could not find anything that wasn't working, so I included the code in my project which gave really good results (see figure 3).

(a) Original buffer, showing distortion.



(b) Working buffer after including the fully geodesic buffer.

Figure 3

7

HOW THE APPLICATION WORKS

7.1 USERS

To be able to use the application you need to create a user, or log in if you have a user already. You will be redirected to the projects-view, where you can see all of your projects. You can add and delete projects, and the progress of each project is saved automatically when changes are done.

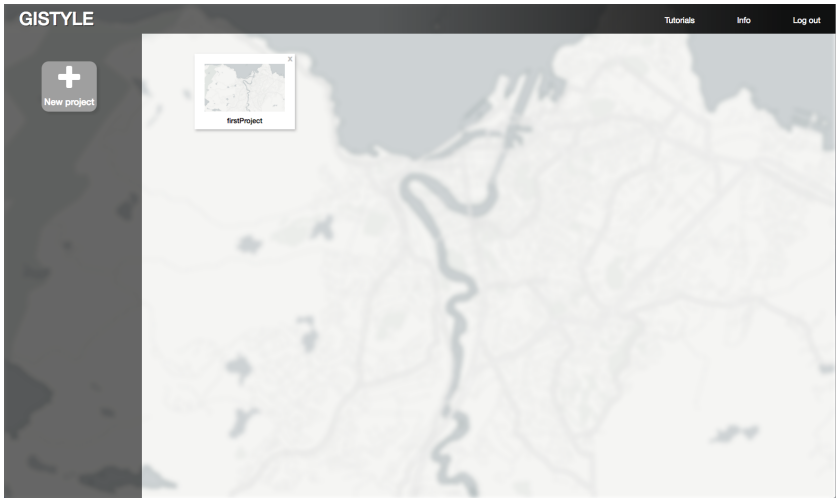


Figure 4: View of your projects.

By clicking on a project, you will be redirected to the main page of the application which contains all the tools and the interactive map.

7.2 LAYERS

When a new project is created, some default layers are automatically added to the project. This way you can easily start working with the tools and learn how the application works without having to upload your own data.

Layers can be dragged and dropped in the layer menu to change the order they are drawn. You can delete layers, add new ones (so far only GeoJSON) and change styling. You can also draw new layers by clicking the polygon icon in the bottom right corner.

7.3 TOOLS

The tools available in GISTYLE are:

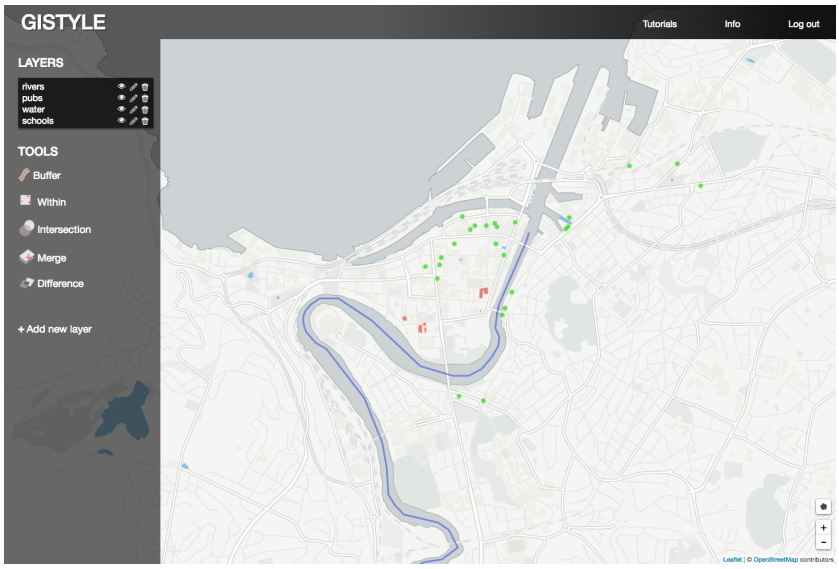


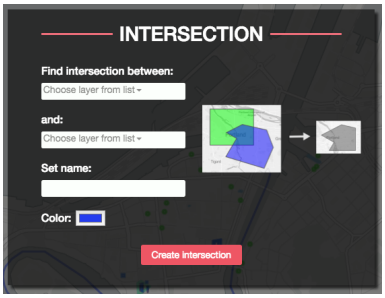
Figure 5: View of how a new project looks when opening it for the first time.

- Buffer
- Within
- Intersection
- Merge
- Difference

How each of these tools works are explained in the tutorial section of the application. Most of them work the same way as in other GIS-applications, but the difference between ”Within” and ”Intersection” should be clarified.

”Within” returns all elements in the input-layer that has parts, or all of the element, inside the input-area. ”Intersection” on the other hand, will return just the part of the elements that is intersecting. ”Within” also handles both points and polygons, while ”Intersection” only handles polygons.

(a) Pop-up window for within



(b) Pop-up window for intersection

Figure 6

The implementation of each of the tools are implemented on the server side and can be found in GISTYLE/routes/api.js. Most of the tools are implemented by simply using the turf library, but some also had to be tweaked a little bit to get

the desired functionality. An example is the within-function. Turf has a method called ”turf-inside” that I used for this implementation, but it only handles points. Since i also wanted within to handle polygons, I had to add some extra code to also check if any of the points defining the polygon was within the area.

8

DISCUSSION ABOUT THE USER INTERFACE

When creating a web application one of the most important parts is that the app is intuitive. It doesn't matter how fast the algorithms are, if the user doesn't understand how to run them. I therefor spent a lot of time prototyping the user interface using sketch (see 4.5).

I decided to use a basic side menu and top navigation bar for the main page of my application. Since menus often take up a lot of space, I decided to make them see through to increase the size of the map that would be visible. Black was also a natural choice of color so that the menus would intervene as little as possible with the design of the map.

I also wanted to use a basemap with as little detail as possible, so that the user could easily distinguish between the basemap and the data layers on top. I therefor chose one of the basemap's from Mapbox with as little detail as possible.

One of the main ideas with the user interface was that it should be intuitive how to use the GIS-tools.

When deciding how to implement the interface for different tools, like the "Within" tool, I thought about what information would be important for users that was unfamiliar with GIS. An important part here, was that I wanted explanations for what the different tools actually calculates and outputs. I therefor decided to implement the tools as pop-up-windows, or as often called: dialogs. A pop-up-window, instead of for example a drop-down-menu underneath each tool, would make this easier and more intuitive. I wanted the explanations to be as visual as possible, and I therefor made figures explaining the inputs and outputs of each tool (see figure 7 for an example).

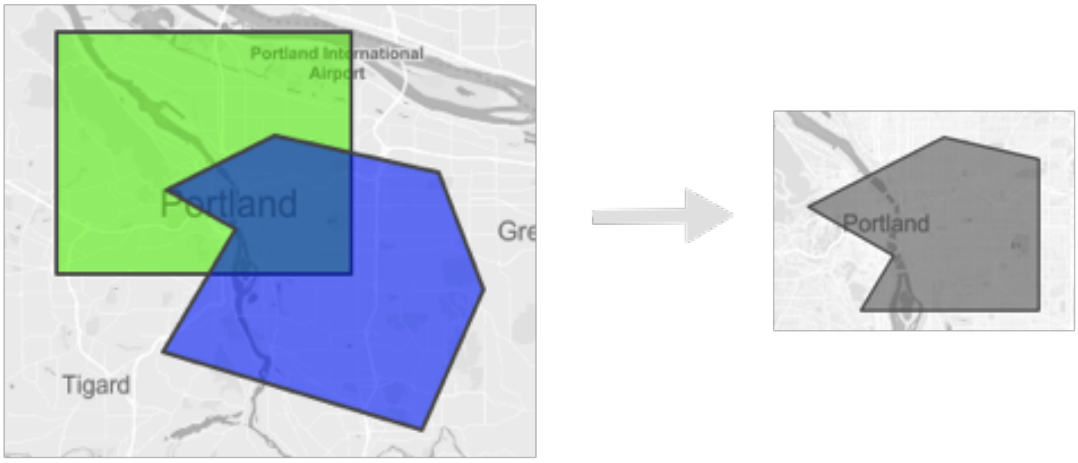


Figure 7: Example of figure used to explain how tools work

Another important part of the interface was to make it as difficult as possible to let the users make mistakes. For example will drop-down menus only contain the type of layers that is supported for the current tool. Another example is that when uploading a layer, a check to see if the layer is actually a GeoJSON, is done.

9

CONCLUSION

9.1 DOES THE APPLICATION WORK AS PLANNED?

I managed to implement the user functionality as planned. All progress is saved to the database, passwords are encrypted and user are authenticated for the different pages. You cannot enter your projects without having an active connection for the specific user you are trying to connect with.

I'm also pleased with the user interface. I have tested it on users, and they can easily navigate through the application and use the tools as planned.

The functionality I wanted is also supported, both the tools for doing spatial-analysis, and also support for changing the styling of different layers. Important functionality, as for example being able to reorder in what order layers are drawn to the map, also increases the value of the application.

9.2 DID I MAKE GOOD CHOICES TECHNOLOGY-WISE?

I'm happy with most of the tools I chose to use. The mean stack worked well, especially MongoDB, since most of what I'm saving is GeoJSON. Angular, on the other hand, I never really got comfortable using and I instead wrote big parts of the application in pure JavaScript. If I had to chose the framework again, I would have gone for TypeScript with react instead.

A challenge with using this type of technology which is quite new and constantly changing, is that it is difficult to find tutorials that is not outdated.

When it comes to the spatial analysis, I wish I would have switched to turf earlier, and not used as much time on PostgreSQL and CartoDB.

9.3 FURTHER IMPROVEMENTS

There are many further improvements for GISTYLE. One of them is to increase the responsiveness, and to do more testing for different browsers and operating systems.

There are a lot more functionality I would have liked to add to my project. You're never really finished which a project like this, and there are endless of functionality that would have been fun to implement.

I would have liked to allow uploads for more than just GeoJSON. There are many libraries that converts to GeoJSON, so it wouldn't take to much time to add this.

One important functionality that is missing now, is intersection for lines. This is not supported by turf yet, but it will probably be added to the library soon. There is also a problem with buffers on lines, that create overlapping buffer zones

because of the round edges. I could not find a solution for this unfortunately. I would also like to add more functions for spatial analysis. Clip, dissolve and distance are some examples.

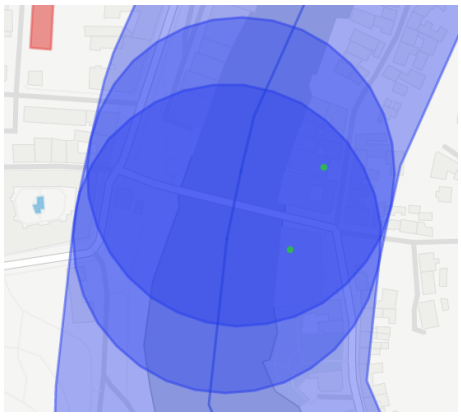


Figure 8: Ends of buffer are round and causes overalapping buffer zones

I would also like to increase both interactivity with the map, and the amount of information the user can get about the layers. I would like to implement functionality so that the user could click on a layer, and get information about the layers, for example the information saved in the properties attribute.


```
"properties": {
    "@id": "relation/5258705",
    "alt_name": "Kyvannet",
    "name": "Kyvatnet",
    "natural": "water",
    "type": "multipolygon"
},
```


Figure 9: Example of properties attribute for a GeoJSON

9.4 RESULT

The web application can be found at *gistyle.herokuapp.com*, and all the code can be found at *https://github.com/mathildor/GISTYLE*.

BIBLIOGRAPHY

[1] Heroku, *Salesform.com*, 2016 :
https://devcenter.heroku.com/

[2] The mean stack, *Chris Sevilleja,* 2014:
https://scotch.io/tutorials/setting-up-a-mean-stack-single-page-application

[3] MongoDB, *Tutorials Point* ,
http://www.tutorialspoint.com/mongodb/

[4] Angular.js, *w3schools*
http://www.tutorialspoint.com/angularjs/

[5] Angular.js, *tutorialsPoint*
http://www.w3schools.com/angular/angular_intro.asp

[6] Node.js, *Christopher Buecheler* 2015:
http://cwbuecheler.com/web/tutorials/2014/restful-web-app-node-express-mongodb/

[7] Express, *Node.js,* 2016:
https://expressjs.com/

[8] Leaflet, *Vladimir Agafonkin,* 2015:
http://leafletjs.com/reference.html

[9] JSTS *Björn Harrtell*
https://github.com/bjornharrtell/jsts

[10] Turf, *Morgan Herlocker* 2016:
turfjs.org

[11] Turf buffer, *Morgan Herlocker* 2016:
https://github.com/Turfjs/turf-buffer

[12] Geodesic buffer, *Manuel Claeys Bouuaert* 2016:
https://github.com/Turfjs/turf-buffer/pull/33

[13] Turf vs JSTS, *Morgan Herlocker,* 2013:
www.morganherlocker.com/post/turf-vs-JSTS/

[14] GeoJSON validation *null_radix*
https://www.npmjs.com/package/geojson-validation

[15] PostgreSQL, *The PostgreSQL Global Development Group* ,
https://www.postgresql.org/

[16] CartoDB, *carto*
https://carto.com/builder/

[17] proj4js, *Jason Long*,
http://proj4js.org/

[18] Passport,
http://passportjs.org/docs

[19] Sketch, *Bohemian BV*, 2016:
https://www.sketchapp.com/

[20] Trello, *Trello, Inc*
https://trello.com/guide

[21] Web mercator, *Spatial Reference*
http://spatialreference.org/ref/sr-org/7483/