Universitetet
i Stavanger

DET TEKNISK-NATURVITENSKAPELIGE FAKULTET

BACHELOROPPGAVE

Study programme/specialisation: Bachelor in Computer Science / Datateknologi	Spring semester 2022 Open
Author(s): Fatima Matieva, Mussa Banjai, Ole Dybedokken	
Programme coordinator: Erlend Tossebrø Supervisor: Erlend Tossebrø	
Title of bachelor's thesis: Visualisering av geografiske data som varierer over tid Engelsk tittel: Visualization of geographic data that varies over time	
Studiepoeng: 20	
Emneord: map, ssb, weather,visualisering	Sidetall:72 + github Stavanger 15. mai 2022

Contents

Chapter 1

Abstract

This task creates a web application that visualizes data that varies geographically and over time on a digital map. The development used the PERN as a technology stack and was implemented accordingly. The application uses Mapbox as a digital map, which also has great navigation options in terms of usability. The visualization dealt with data from Statistics Norway and MET which gave real values that were visualized over Norway. Several principles from both User-Centered Design and Visualization Pipeline were practiced.

Chapter 2

Introduction

The project assigned is to develop an application that can visualize data that varies geographically and over time on a digital map. The application can use data from legitimate organizations such as the Norwegian Climate Service Center, traffic data from the Norwegian Public Roads Administration, or data on air pollution from the municipalities. An attempt would look at how the maximum and minimum temperature for a given period, for example, a month, has varied geographically and over several years. Our approach is to use data that will be selectively advantageous for the application as the aim is to expose it to a larger group of people. Much of the data can be challenging to comprehend, and thus we must consider it attentively. In addition, there is a lot of data collected through data analysis across virtually all fields that are difficult to interpret; therefore, we will select data sources accordingly.

A graph or map that gives the data a visual context makes it easier for the human brain to interpret. Developing an application that will incorporate the intention is vital and comprehensive. The overall purpose of any form of visualization is to deliver data in the most efficient way possible. Ultimately, one must go in-depth to identify areas that should be considered and comprehend the overall field to attain the desired goal. The purpose of visualization is to be perceived. A successful visualization aims to convey information efficiently and accurately to the targeted audience while fulfilling the objective of visualization. It is common for a visualization application to display multiple visualizations simultaneously, resulting in various mappings and renderings of representations. Each data set can be mapped to graphical entities and attributes in many ways. Accordingly, several interactive tools can be provided to the user.

As part of creating an application where users can interact with visualizations, developers must decide, among other things, the mapping of graphical attributes to data fields, modify views by selecting and implementing methods, and determine how much data to visualize. To ensure that interpretation is made as easy as possible, additional data and mapping information are necessary, and they must be integrated into the visualization. A final consideration is the aesthetics of the resulting display, which will emphasize user interface and user-centered design principles. The user interface elements are components that can enter, present, explore, and measure data. The user interface components are often entered through dialog boxes; they might be visually represented for selection ease. For simplicity and precision, this development will be presented through a pipeline[11].

The website is presented at: Github, Website
Demo video of the website: Demo Video

2.1 Project Scope

In order to develop the predicted application, research must target the expected scope. Data sets with geospatial attributes are the only ones suitable for use since they provide location information on the matter. It is thus necessary to adapt to such data fundamentals and understand what a successful visualization requires. Moreover, successful visualization also means the target group can effectively perceive the data and benefit from it. This conquest requires developing an application at the depth necessary to meet the needs of its users. For the application to benefit the user, a thorough understanding of user-centered design principles must be considered. The user-centered design ensures that we examine how effectively an application achieves its designed purpose.

2.2 Objectives

Having the goal and background of the task defined, the application can be designed accordingly. As the main objective of the thesis is to apply the fundamentals of visualization to visualize valid data through user-centered design principles, the composition is as follows:

- Develop a complete web application with consideration to the user interface.
- The application uses a digital map for visualization.
- The data used is real and gives a position as to where it occurred.
- An interface where user can choose between data and visualization.
- Visualization takes place based on research and the visualization pipeline.
- The overall application considers user-centered design principles for the benefit of the user.

2.3 Thesis Outline

The following chapters will constitute this thesis

- **Chapter 2** Background information that provides a general understanding of what we are attempting.
- **Chapter 3** An overview of several requirements needed to achieve visualization.
- **Chapter 4** An outline of how the web application is built considering other technology options.
- **Chapter 5** Following is a formation of a pipeline that was used to visualize data.
- **Chapter 6** The description of user-centered design principles and key aspects for fulfilling a complete user foundation.
- **Chapter 7** A review of the prototypes created and key features of the functionalities created accordingly.

- **Chapter 9** A business view of what an application like this could cost and profit.
- **Chapter 10** Conclusion and future work

Chapter 3

Background

Raw data becomes valuable only when we apply methods of deriving insight from it. Since humans are intensely visual creatures, we must visualize to communicate faster. This can be done in several different ways, and in the following chapter, we will explain what inspired us and what was used for implementation.

3.1 Web Application

Instead of a stand-alone application, a web application was chosen to reach a larger audience. By using web-based technologies, anyone can access and interact with the application. In addition, it will be possible to use regardless of operating system and device type[11].

3.2 Similar Products

In the following chapter, similar products are presented as inspiration. They were used as a blueprint when implementing the application. Hence, the functionalities and appearance of these sites influenced the end product.

3.2.1 Windy

Windy is a website that provides a weather forecasting service. Windy has many features such as European weather radar, thunderstorm forecast, and satellite overlay. Windy weather became a primary source of weather information for governments, institutions, and individuals in affected areas during the hurricane in 2017. One of the reasons Windy inspires the project is how the map looks regarding the temperature. The colors and the effect look amazing. Also, the range of options that the website provides on the quick menu is impressive, such as weather, airports, radiation, and requirements.

3.2.2 Meteoblue

The Meteoblue website delivers local weather information worldwide for any point on land or sea. One thing that brings attention is how clear it is to see the difference in, for example, weather, temperature, and other features. A box with all the information appears when a specific city is selected.

3.2.3 Ventusky

The Ventusky is a web application that has been developed by a Czech meteorological company called InMeteo. The whole focus of this application is to provide weather predictions and meteorological data visualization. One of the cool features of the application is the ability to switch units, such as Celsius and Fahrenheit, when it comes to temperature, or k/m and m/s for wind speed. Another reason is that when we click on a specific city, the full info is shown as a sidebar. The help bar is also something to look up because it is straightforward to read and understand the application's details and functionality. The downside of this app is the sensitivity of zooming in/out the map, making it difficult for the customer to get to the place he wants. The app can also be buggy, which can contribute to user's experience negatively. Also, the help page is unclear, as it should be also a guide about how to use the app, rather than just Q&A.

3.3 Target Group

It was intended to make the application general benefit a broad target audience. This means that the application must be so simple that even a pupil can use it in a school context, and an elderly person can use this. Both SSB and Frost are providers of general data that can interest anyone. The application has the main focus of being easy to navigate through but also fully shareable. Small startup media companies can also benefit from a site like this as it helps give a clear understanding of data but also good-looking visualization as a GIS specialist might be expensive. Graphs are downloadable, and maps are fast, clear, and easy to visualize.

3.4 Software Background

The following chapter will explain what software tools were used to make the visualization application. The research was conducted early in the development process to select the following tools. Because this is a web application, popular programming languages, libraries and plugins were used in the development. An explanation for the use of these plugins can be found in chapter 4.5

3.4.1 PostgreSQL

PostgreSQL was the database system of choice. PostgreSQL is an open-source object-relational database management system focusing on extensibility and standards compliance. PostgreSQL was initially developed in 1986 as a follow-up to INGRESS. PostgreSQL has, in recent years, risen in popularity and gained a more extensive community with this and is one of the more popular choices among the relational database options.

3.4.2 Node.js

Node.js is an open-source runtime that lets developers use JavaScript to build I/O applications which are server-side driven. Due to fast building, scalability, and efficiency, many applications are built on it. In this project, Node.js was the choice for building the API back-end service for several reasons.

3.4.3 Express.js

Express.js or Express is an open-source framework that sits on top of Node.js. It was created to simplify Node API and add helpful new features. The framework helps with routes and requests. Express was used to speed up the building time and take advantage of its simplicity. Express gained popularity due to its use in the technology stack MERN and MEAN.

3.4.4 Postman

Postman is a tool that helps to build, test and modify APIs. This tool is used to create and save simple and complex HTTP/s requests (such as GET, POST, DELETE, PUT and more) as well as read their responses. It helped for the group to be more efficient by creating those tests without spending too much effort on coding scripts. This is done by allowing users to create and save simple and complex HTTP/s requests, as well as read their responses. The result - more efficient and less tedious work. It was crucial for this project since it helps to see response from requests calls, without affecting the code.

3.4.5 React.js

Reactjs became the front-end web development utility of choice. React is a JavaScript library used to build single-page applications. ReactJS is based on a subsidiary language of JavaScript, called JSX, which lets developers embed HTML-like code into JavaScript code to create interfaces.

MUI

MUI was were used for a variety of reasons discussed in chapter MUI. MUI (Material-UI) is a library of UI components used to creates web applications faster. MUI offers already built components that are made only for React.

React-Query

React query is a React library that simplifies how React fetches, caches, and synchronizes data from the server. It is often looked at as the "missing data-fetching library" for React. It makes updating the server state in React's application much easier, as this is something React is often criticized for.

3.4.6 Visual Studio Code

Visual Studio Code, known as VS Code, is a source-code editor created by Microsoft. VS Code includes many features, such as debugging, syntax highlighting, snippets, intelligent code completion and more.

3.4.7 GitHub

GitHub was used as a work platform for version control of the application. This became a natural choice, since it was used often during academic years. GitHub is an online software development platform used for storing, tracking, and collaborating on software projects. It gives developers the ability to upload their own code files and to collaborate with their fellow developers on projects.

3.4.8 pgAdmin

pgAdmin is a web-based GUI tool used to interact with the Postgres database sessions, both locally and on remote servers. pgAdmin performs any database administration required for a Postgres database. pgAdmin is also an excellent tool for spatial data as it lets the user visualize different points on a map stored in the table. [1]

Chapter 4

Requirements

An important part of web application development is the requirements gathering process. As the purpose and integrations the application wishes to include are clearly stated, we must research and determine the necessities to help solve the task. Several requirements were identified early on, and decisions were made consecutively. Other requirements were not determined until they were implemented, and the explanation of choice lies in the development area. The following topic will provide the requirements needed to ensure development is achievable.

4.1 Geographic Information System

The first step in creating a visualization is determining what data will be displayed. Considering that data is collected from multiple sources, we have to decide which ones are most legitimate and will benefit the application. The project description specifies that geographical data varying overtime is requested to be visualized, so it was necessary to obtain data that was able to exhibit this. An ideal example of data is statistical data and weather data visualized geographically. This type of data is geospatial data, which refers to data that is given with a specific location within the real world. An attribute of such data would also be a timestamp, which identifies a time, date, and year through an ordered sequence of characters. Identifying data to be visualized is specified described in chapter 6.

Since geospatial data will be visualized, a database will be necessary to accommodate such data. Geospatial data represent objects or features on Earth's surface to visualize places, events, and elements. The best way to store geospatial data is using a database system that supports a geographic information system (GIS). With GIS, data and maps interconnect, enabling the integration of location data with various types of descriptive information. GIS includes positions and geometry to specify the location. The most significant advantage of the system is that it improves performance. Comparing a spatial database with a non-spatial database, the former stores and retrieves both spatial and attribute data, whereas the latter allows only attribute data to be stored and retrieved. What distinguishes GIS from other database systems is its geometry column with data in a specific coordinate system defined by spatial reference identifier (SRID). A description of how GIS was used for visualization is provided in 6.

Another way of storing geospatial data is using a GeoJSON file. A GeoJSON file contains both attribute data and geospatial data. Kartverktet.no is Norway's national mapping agency. They have, in recent years, only offered their own format SOSI for administrative units but have, for the last few years, also offered GeoJSON due to its rise in popularity. GeoJSON is very popular for development since it is easy to use and can save files directly.

4.1.1 Example usage of GIS

The main reason for using a database extender like PostGIS is that it allows location queries to run in SQL. A quick example of it would be the ability of S_AsGeoJSON to convert points stored in a table to a GeoJSON file format. In this chapter, an example will be shown of how several PostGIS functions were used to find the convex hull of all stations that Frost offers.

To interpolate the different weather values available for a specific time and only for Norway, it was needed to have a region of interest for each particular time period, so it doesn't interpolate over the whole map and confuse the user. Some stations may offer a mean temperature for January 1985, and others might not, so the points will differ for every period. This example constructs a query that will find the region of interest of Frost's available stations. The lat and long of all the points were known from the API, as explained in **??**.

To find the convex hull, PostGIS needs the points stored in a GeometryCollection. Using ST_COLLECT, all the points stored as geometric type Point in the sources table are gathered into a GeometryCollection. pgAdmin offers a Geometry Viewer tool that lets users display the results of queries on a map. The screenshot below is a screenshot displaying the query

```
SELECT (ST_DumpPoints(ST_Collect(geog::geometry))).geom FROM sources;
```

The ST_DumpPoints turns it back to Points so it can be displayed and visualized on a map in pgAdmin.

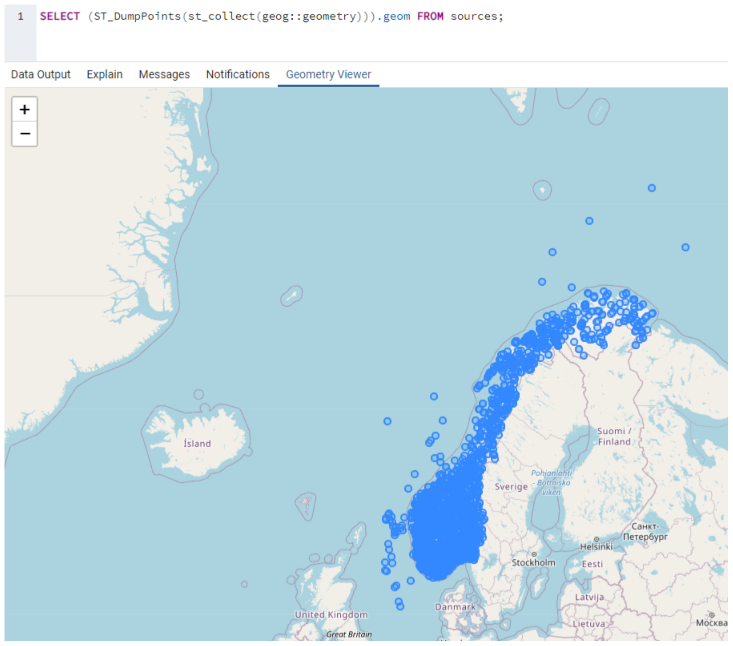
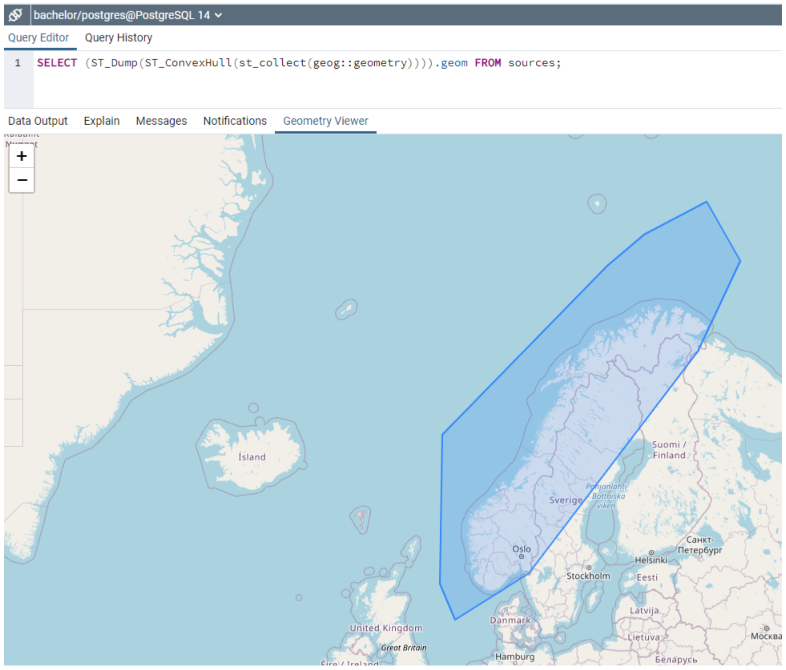

```
SELECT (ST_DumpPoints(st_collect(geog::geometry))).geom FROM sources;
```

Figure 4.1: Display all sources in Norway.

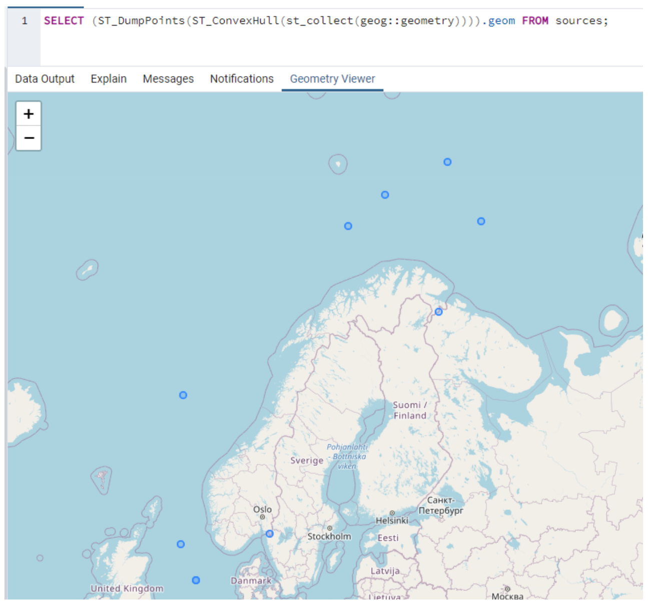
To create a polygon representing the smallest convex geometry that encloses all geometries in the input ST_ConvexHull function can be used.



```
SELECT (ST_Dump(ST_ConvexHull(st_collect(geog::geometry)))).geom FROM sources;
```

Figure 4.2: A polygon representing the most outer points.

This is only done for demonstration purposes to understand better what happens when ST_ConvexHull is run. Replacing ST_Dump with ST_DumpPoints will return points representing the region of interest.



```
SELECT (ST_DumpPoints(ST_ConvexHull(st_collect(geog::geometry)))).geom FROM sources;
```

Figure 4.3: The most outer points out of the all the sources.

Combining this with the other tables and limiting it to a specific time and element will return the needed region of interest.

```
SELECT ST_AsText(
(ST_DumpPoints(ST_ConvexHull(ST_collect(geog::geometry)))).geom)
FROM(SELECT geog FROM sources s
INNER JOIN weather w on w.source_id = s.source_id
INNER JOIN weather_data d ON w.weather_id = d.weather_id
WHERE d.time ='2019-01-01' AND d.element ='mean(air_temperature
P1M) As x;
```

4.2 Map

Digital media is an integral part of modern visualizations. Since the goal is to visualize geographic data, it is a must to have a map. The two most known map APIs are Google Maps and Mapbox. Below will explain each of the map

APIs.

4.2.1 Mapbox

Mapbox is a location data platform that powers the maps and location services used in many websites and mobile apps. Many well-known companies use Mapbox services, like DHL, Shopify, Airbnb, etc. Mapbox relies on Node.js

Advantages:

- **Customization and flexibility:** Mapbox Studio and Mapbox API give vast freedom to explore the customization. Adding objects, choosing layers to be displayed, changing colors, and changing fonts can be done effortlessly.
- **Performance:** Loading data can take time, so it is crucial to have a map that reduces loading time. Thanks to the optimization of Mapbox GL JS, it is expected to not only load quickly but smoothly.
- **Open-source:** Mapbox encourages the community to inspect its code and improve it. It also has a library called Mapbox GL Native that allows customizing interactive maps into apps.
- **Offline mode in API:** Mapbox supports this functionality because of vector maps use.

Disdvantages:

- **Weak coverage:** Mapbox relies on OSM(OpenStreetMap), making some regions not updated, for example, India and China.

4.2.2 Google Maps

Google Maps is a Web-based service from Google that provides detailed information about geographical regions and sites worldwide. Snapchat and Accenture are one of the companies that use Google Maps.

Advantages:

- **Best information:** Due to Google's satellites, Street View vehicles, and user-generated corrections.
- **Street view:** Street view provides interactive panoramas from different angles on the streets worldwide.
- **Extensive language support:** It supports 80+ languages, and it is still growing.
- **Recognizable:** Most of smartphones and computers are familiar with the interface of Google Maps.

Disadvantages:

- **Few options for customization:** Google Maps API is limited when it comes to options for creating a unique-looking map.

4.2.3 Map of Choice

Our goal is to visualize data on a map simply. With that being said, Mapbox seems to be a better option for us since it gives a range of customization, has excellent performance, and it is open source. Moreover, since the focus will only be on Norway, the weak coverage does not affect our project. Mapbox also offers easy integrations of tools like leaflet and deck.gl. This might be advantage as they are great for visualisation of data.

4.3 Database

The database was created with PostgreSQL. The main functionalities of the database are to provide data and storage to the application. The database is divided into four tables and all of them are related to the weather part of the application. The table *sources* contain all the details and information about the different stations located in Norway. The *weather* table stores information about what sources offer what type of date and from when. The table *weather_data* stores the actual values for the weather. The *weather* table is used as a link between sources and *weather_data*. This makes the information about the data more accessible. The elements could have been linked with the *weather* table but it was not necessary.

The code that creates the tables is built in the *Server/tools* folder and exported into *Server.js*. *Server.js* is created in a that will allow for an admin page to be easily created in the future. With some small adjustments to the code a post request can be sent to /api/v1/getWeatherDataForSource with an element as a parameter and if the element does not already exist the database will automatically implement data for that element fetched straight from the frost API. As the main object of this application was the visualization part we decided not to integrate the admin page however the back-end is set up in a way that makes this possible easily.

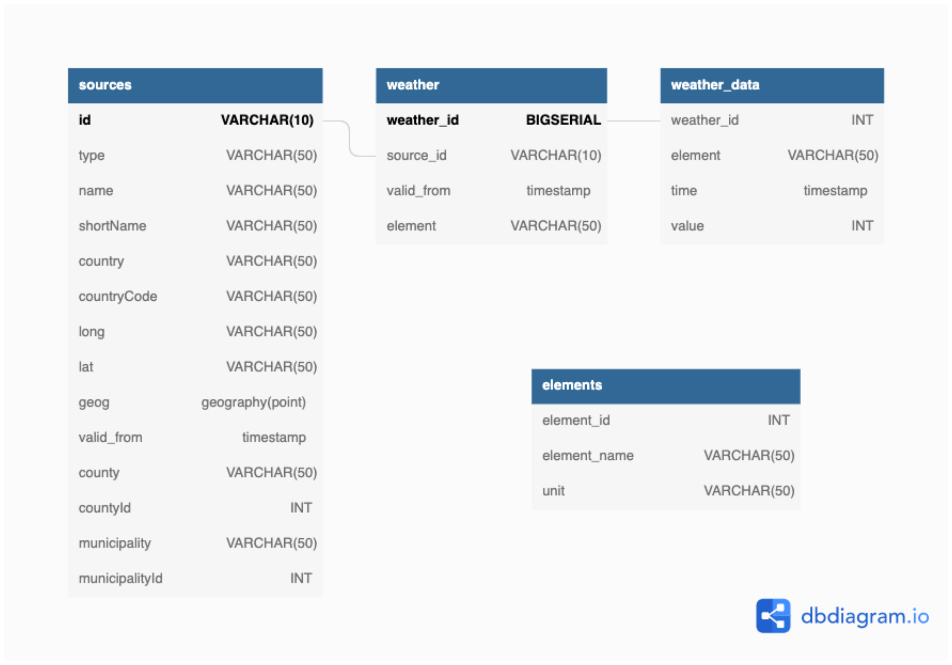


Figure 4.4: The database schema.

4.3.1 Sql Injection

Although the web application does not require any login, it is quite important to protect our data from any type of attack. SQL injection could be one of the potential attacks. An SQL injection is when SQL code-snippets are inserted into fields that communicate with the database using SQL. Common examples of such fields are login and registration fields. What must then be determined is which input field to insert SQL snippets into. By avoiding any SQL injections, we kept data separate from commands and queries.

Chapter 5

Web Application Architecture

An application's architecture defines the interaction between its applications, middleware systems, and databases. In light of previous experience, we intended to find a technology stack that would benefit us and the development of the application. Technology stacks, also referred to as solution stacks or software stacks, combine various technologies used to build and run one application. Technologies include programming languages, frameworks, databases, front-end tools, back-end tools, and applications connected via REST APIs. The two primary components of an application are client-side and the server-side.

Choosing a stack simplifies the build time and enhances productivity. Selecting the right stack is paramount to what kind of product will be built and how efficient it will work. Therefore, the success of a project depends on the technologies used. In order to implement a stack, one must take into account several segments. First and foremost, one must consider why it would be beneficial to use a stack. Secondly, it must be evaluated if the stack will fulfill and meet the set requirements. Other considerations include whether the whole stack or only parts need to be utilized. A critical aspect of web development is to evaluate the different technologies that can be used.

The following chapter will view the selected stack, how it meets the set requirements, and provide alternatives that can be implemented instead. The alternatives presented are based on the popularity of usage and were also introduced during the course of study. The choices were all considered with the thoughts of reaching a stack in the end. The amount of stacks out there are almost unlimited but some are more popular than others.

5.1 Considering Databases

Choosing the right database can save a lot of time. In order to choose the right one, there is certain points that has to be considered. The first criteria to find the most fitting database is to figure out if a SQL database or a NoSQL database is needed, or maybe both. This first criteria will have an impact on next criteria, which is choosing the database management system.

5.1.1 NoSQL vs SQL

SQL, relational databases, is one of the most used and has the most significant community but can be quite restrictive due to schemes. NoSQL does not require a set structure and with this is more flexible. When comparing PostgreSQL to MongoDB, which would have been the two of choice, PostgreSQL got measured 4-14x faster than MongoDB in transaction performance in a benchmark test provided by OnGres in late 2019.[12]

SQL	NoSQL
Advantages • Bigger community • Better for multi-row transactions • Faster for joins and queries • Experience • Faster in complex queries	**Advantages** • More flexible • Easier to scale • Mainly used for data analysis • Rapidly growing
Disadvantages • Forced to follow a schema • Scales vertically (Increasing RAM, CPU, SSD or other hardware performance)	**Disadvantages** • Lacks complex query functionality used in SQL. • Haven't been around for long so might lack community

Due to the fact that the majority of the data were historical, we concluded that changes would rarely occur. Most historical data is stored in rows and columns, fitting for a relational database.

5.1.2 Final choice

The decision came down to the experience we had with SQL. We had no experience working with NoSQL databases. We did not find any significant disadvantages to it other than that working with JSON APIs is a better option to use NoSQL since it is more flexible with data types. Nodejs offers some great libraries that simplify the conversion from JSON to SQL so this solves that problem. SQL is also best suited for working with complex queries, which fits our project. Queries make it possible to perform actions like join.

The benchmark performance test by OnGres also clearly shows that SQL seemingly was the better option mainly due to experience, community, speed, and complex queries. The only disadvantage was that it limited us to use schema, and in the future of this project, scaling might be a problem or more expensive at least.

5.1.3 PostegreSQL VS MySQL

Working with maps means working with geospatial data. Geospatial is a term used to describe any data related to or containing information about a specific location on the Earth's surface[7].

Most SQL databases offer extensions for this type of data. PostgreSQL, which is one of the largest growing databases, offers PostGIS. PostGIS is one of the most well-known and complete spatial databases around. An example of a feature provided by PostGIS can be ST_CLosestPoint which can find the point closest to a line or polygon. [2]

There are several RDMBS options out there. We had experience with primarily SQLite and MySQL. Because a spatial database could be handy in a project like this, PostgreSQL also had to be considered. This chapter will mainly compare MySQL and PostgreSQL as SQLite was out of the discussion quite early due to the lack of data types.

MySQL and PostgreSQL are both open-source, but MySQL is corporate through Oracle while PostgreSQL is unincorporated. PostgreSQL is an object-relational database management system, while MySQL is a relational database management system. In short, this means that PostgreSQL offers more complex data types and allows objects to inherit properties. MySQL does have a more significant community, but PostgreSQL is rapidly increasing. In the Stack overflow survey of 2021 it showed that 50 percent of developers are using MySQL[5] while PostgreSQL only had 40 percent. They both offer a good documentation.

MySQL offers less complex queries and is seen as easier to learn. The benchmark tests tends to come with contradictory results. However, PostgreSQL is generally faster when dealing with massive datasets, complicated queries, and read-write operations. On the other hand, MySQL is faster for read-only commands. [3]

We have prioritized community and experience in our technologies choices until this point, but we made an exception in this case. We wanted to have the option of storing more data types and performing more complex queries. In the spatial database community, it seems like PostGIS is superior. PostgreSQL came out faster dealing with massive data sets, which could be the case with a web application that depends on data over time. They do offer some different types of indexing; however, they both offer the standard types. With this in mind, we decided to go with PostgreSQL.

5.1.4 The final choice - PostgreSQL

The PostgreSQL database management system is an open-source object-relational database management system focusing on extensibility and standards compliance. This system can manage even the most demanding data workloads securely and safely by using and extending the SQL language. With ACID compliance, PostgreSQL guarantees data integrity regardless of errors, power outages, and other mishaps. Data is stored in a tabular format and interconnected by constraints, triggers, roles, procedures, and views. PostgreSQL's many features are PostGIS, an extension that provides a spatial database. It allows location queries executed in SQL to run on geographic objects.

Spatial data types such as distance, area, intersection, union, and specialty geometry are added to PostgreSQL by PostGIS. As PostGIS is a spatial database, the data includes a geometry column with data defined using a spatial reference identifier (SRID). In addition to storing geospatial data, it also supports non-spatial data since it has all the standard PostgreSQL database functionality.

5.2 Front end utilities

The front-end part of the application is expected to create a user interface where the user can navigate and communicate with the page. By default, a website's structure and content are laid out through HTML, styled using CSS, and advanced interactive features enabled by JavaScript. The use of libraries and frameworks built on these programming languages makes the development process differ. It is imperative to consider several factors when choosing a framework or library.

This web application aims to display visualizations on maps, so the front end will concern the user experience. Charts and maps are most likely to be taken care of using already made libraries. It was decided to use JavaScript libraries for the front-end, as most of the website comprises JavaScript (around 95%). Also, it fits most web browsers. This chapter will cover and evaluate the three main JavaScript front-end libraries or frameworks.

5.2.1 React

React is a free and open-source JavaScript library used to build single-page applications. ReactJS is based on a subsidiary language of JavaScript, called JSX, which lets developers embed HTML-like code into JavaScript code to create interfaces. React is used to create single-page applications, and it features a wide variety of libraries created by other developers in its community. React allows the creation of reusable UI components, making code easier to understand.

The more popular choice among developers for web development is React, which has a large developer community. According to the 2021 Stack Overflow Developer survey, 40.0% of developers had recently worked with React. In this sense, the more extensive the community, the greater the resources and help available. In addition, React offers users flexibility and makes the components reusable. Especially for a project such as this, where plotting, tables, and graphs are integrated at multiple points, this can save a great deal of time. In terms of updating technologies, React is very consistent. However, it comes with the consequence of a lack of proper documentation.

5.2.2 Vue

Vue.js is a front-end JavaScript framework used to build web interfaces and one-page applications. Vue builds on top of standard HTML, CSS, and JavaScript with an intuitive API and outstanding documentation. As a compile-optimized rendering system, Vue is highly reactive and rarely needs to be manually optimized. In addition, it provides a system that scales between a library and a full-featured framework.

The group has some experience with Vue and that could have been an advantage for this project since it wouldn't be necessary to use the time to learn the framework. However, Vue is limited when it comes to re-render all the components upon a state change. According to the 2021 Stack Overflow Developer survey, only 18.97% had worked with Vue, which can be challenging when it comes to obtaining resources and help.

5.2.3 Angular

Angular is a platform and framework for building single-page client applications using HTML and TypeScript. It has many features such as two-way binding, dependency injection, RESTful API, and AJAX handling. A problem with Angular is that simple and small applications can become bloated because of their requirements of boilerplate codes, abstractions, and other bundled features.

5.2.4 The final Choice

After considering those JavaScript libraries, the final decision ended up being ReactJS as a front-end technology tool. Reactjs lacks proper documentation compared to Vue, however, it is quite hard to compare as they both have great documentation and most of Vue's early days documentation is written in Chinese. React offers React-map-gl which is a great React API for Mapbox created by the Uber team. The lack of experience was not so impactful, as a group knew JavaScript syntax, and learning JSX is not that steep. Additionally, it is a very good single-page application framework, so transferring the design to other platforms will be a lot easier and smoother.

5.3 Back-end services

As described in the data visualization pipeline chapter in chapter 5 some of the data will need to be reformatted to be visualized. Most of the work regarding data and formatting will be happening on the back-end. Choosing the right back-end services can affect speed, memory, and efficiency. Some applications may require demanding calculations and need more powerful back-end services. In selecting the exemplary back-end service for this project experience, speed and community were the main factors considered.

5.3.1 Python/Flask

The Python programming language is inherently object-oriented, dynamic, and high-level. Python is ideal for scripting with its syntax and dynamic typing, making the language best suited for back-end applications, numerical computations, and machine learning. Flask is a lightweight, extensible python web framework that provides valuable tools and features to help create web applications more accessible. Flask is a minimalistic framework making it quick and easy to build a web app prototype after installation. Flask is not asynchronous friendly, meaning it can only handle one request at a time, making it not very good for real-time applications. Do not mistake this for scalability as Flask is highly scalable since it can handle a large number of requests each day. In short, Flask helps make Python syntax server-side web applications easily. It is highly scalable but is not asynchronous which makes it not great for real-time applications.

5.3.2 Node/Express

Express is a Node framework for developing back-end web applications, providing robust features for mobile and web applications. Express is the most popular Node framework used in popular technology stacks lean MEAN, MERN and PERN. Node can work asynchronously, eliminating the waiting time and continuing with the subsequent request. Single-threaded, non-blocking, and memory efficient are other characteristics of Node. This platform is built on top of Google Chrome's V8 JavaScript Engine, making it fast. Node uses the popular JavaScript syntax resulting in a big community. Node uses NPM, making installation of Express and other libraries very easy. Express is a lightweight, flexible but also minimalistic framework with few dependencies with great and simple routing. A disadvantage to using Express is that it follows no strict standards making it sometimes difficult to maintain code and it struggles with heavy computing tasks.

5.3.3 The Final choice

As mentioned already, community, speed, and experience were the main factors considered in this decision. Node is designed to be publicly accessible; anyone can see, modify, and distribute the code as they see fit. The benefit of Node being open source is the number of resources to use and the active community. The engagement of a community is key to receiving feedback and support for future issues that might occur. Node has more functionalities, while Python is easier to understand. Node is significantly faster than Python because of its non-blocking I/O model and because Node is built on Google Chrome's V8 JavaScript Engine. While Python is suited for developing more extensive projects, it is not the best choice for creating real-time web applications. In the case of experience, the group had some experience with Flask and none with Express itself but considering Express uses JavaScript syntax, the experience in both are equal. This will be a reasonably small project with no need for heavy computations that Python or Node cant handle. The speed and memory will affect the waiting time for the user, where Node has shown to be superior. Due to the speed, JavaScript syntax, and the more significant community Express ended up being the final choice.

5.4 Component library

It is wise to follow a particular theme for the best UI design and not mix several design styles. In this chapter, the different component libraries will be discussed. Choosing a UI component library speeds up the design process and helps take care of many states stored on the front end regarding different styles like dark/light mode.

5.4.1 MUI

MUI (Material-UI) is a library of UI components used to build and develop web applications faster. MUI is highly customizing and is very good for dark/-light mode, languages and different color themes as it takes care of everything when it comes to state regarding that. MUI offers already built components like sliders, navbars, grids, inputs, and sortable data grids made only for React. By having these components already, it saves a lot of time by avoiding long codes from the scratch. Also, it keeps the folder structure more manageable and clean, which makes code more readable. However, to get those libraries, it needs to be installed, and the more libraries are installed, the more it affects the performance.

5.4.2 Tailwind

Tailwind is a CSS framework that comes with predefined classes rather than components as the building blocks of UI design. It gives developers complete control over the styling of a web application. Tailwind does not have a default theme that you have to use like other CSS frameworks. Tailwind building and styling are really fast due to thousands of built-in classes that do not require you to create designs from scratch. The framework was developed by high quality engineers, which is the reason why bugs and breaks do not happen often. Styling and HTML are mixed, which means there is no “separated page” for the styling, and that can be an issue for some developers. Tailwind does not provide certain styling components, such as headers, buttons, navigation bars and so on. This would require creating those features manually. Finally, because it is a really unique CSS framework, it can be difficult to learn it and poor documentation does not help.

5.4.3 Final choice

In the end, the group decided to go with MUI. Since Reactjs is the front-end technology tool that was chosen, MUI looked more logical and suitable for this case. One of the biggest reasons to go for MUI is because it saves us a lot of time from creating a new component scratch and the time that is saved can be used on more complex parts. It also has a huge community, which leads to more resources and help.

5.5 PERN

The final choice ended up leading to PERN. This stack consists of PostgreSQL, Express, React and Node.js. Combining these technologies, one can build a full-stack web application with CRUD operations.

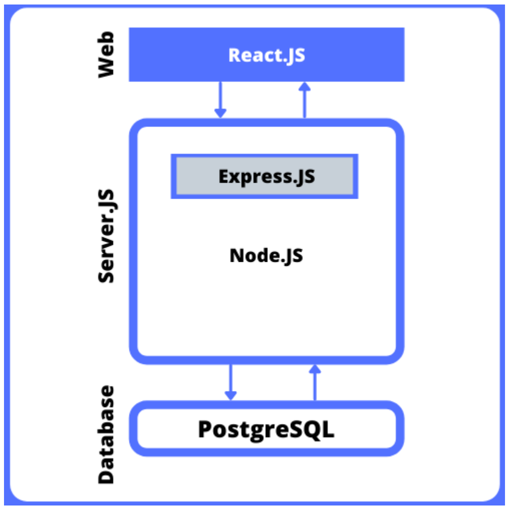


Figure 5.1: Display our technology choices with the help of a PERN stack.

5.6 REST

Rest, or representational state transfer, is an architectural design pattern that defines a set of constraints for creating Web APIs. REST can be explained by the name itself. It means when a RESTful API is called, the *server* will *transfer* to the client a *representation* of the state of the requested resource.[4]

5.7 SPA

SPA (Single Page Application) is a web app implementation that loads only a single web document, and then updates the body content of that single document according to client's requests. It is a huge advantage for both server and client side. The user does not need to load every single time he goes to the new page, meaning that everything happens in one page, as long as they are using the website. This practice makes the website running faster, the user gets a better experience and the caching is more efficient.

Chapter 6

Data Visualization

It is instinctual in our human nature to use sight as one of our primary senses for understanding information. Thus, a graphical representation of information is the easiest and fastest way to comprehend information. This involves mapping data to graphical primitives (points, lines, areas, and volumes), rendering the results, and displaying the results. To grasp why visualization is so important is how fast we humans interpret texts we read; the sequential reading process constrains it[11].

As data grows, static and interactive visualization has become increasingly attractive. Various data sets, including finance, security, and medical, have provided fascinating results that require explanation [11]. Thus, visualization has become a popular tool; there is a growing need to find techniques to communicate information efficiently and effectively. Implementing data visualization will ease users' understanding and interpret structure and patterns by providing an alternative view.

In the sphere of visualization, an recurring drawback is when converting data sets. When sets are transferred from their original format to one that is applicable to the preferred visualizing medium, there can occur issues. Because, in most cases, the conversion of sets is heavily reliant on the tools used to collect the data [9]. Hence, our goal is to aid our users with an easy, and fast process by equipping them with a lot of sample data sets, as well as a comprehensive description of the files formats. The following chapter will describe the visualization pipeline and several entities of visualization.

6.1 The Visualization Pipeline

The visualization pipeline describes the steps involved in creating visual representations of data.

These are the following components of the pipeline:

- Data Modeling
- Data Selection
- Data to Visual Mappings
- Scene Parameter Setting
- Rendering of the Visualization

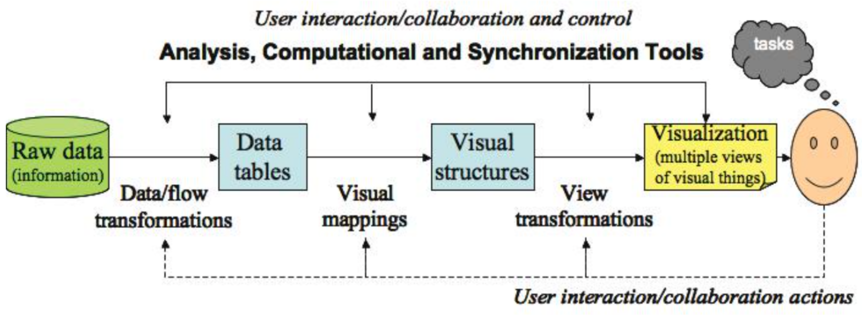

User interaction/collaboration and control
Analysis, Computational and Synchronization Tools
tasks
Raw data (information)
Data tables
Visual structures
Visualization (multiple views of visual things)
Data/flow transformations
Visual mappings
View transformations
User interaction/collaboration actions

Figure 6.1: Raw data is converted into internal representation within the computer to then be display as a visual paradigm.

6.1.1 Data Modeling

The first component of The Visualization Pipeline is data modeling. It is a mechanism to facilitate data access by ensuring its attributes are available in a format that is both quick to access and easy to modify[11]. Initially, the process began by researching and analyzing the type of data available for display. As proposed in the project description, legitimate organizations should be data contributors. That would include organizations such as Norwegian Climate Service Center, the Norwegian Public Roads Administration, and Statistics Norway, just to mention some. The provider would also be substantial for the type of information displayed and conveyed to the user.

Considering the variety of complex and straightforward sources, we decided to use SSB (Statistics Norway) and the Norwegian Meteorological Institute(MET). To begin with, it is imperative to note that the following sources were selected following thorough research at the very beginning of the development process. Statistics Norway is the national statistical institute and primary source of providing insight into the country's social, economic, demographic, and cultural changes. Statistics Norway data is so broad that it appeals to anyone. Research and analysis are also significant aspects of Statistics Norway's enterprise. Norwegian Meteorological Institute offers Frost API free access to MET Norway's historical weather and climate data archive. The data includes quality-controlled measurements of temperature, precipitation, and wind daily, monthly, and yearly. The API also provides additional information, such as metadata about weather stations.

As mentioned in project limitations, general constraints must be taken into account when choosing sources. Both Statistics Norway and The Frost API are free and open-source such that they can be used without license or payment.

SSB (Statistics Norway)

The SSB API can be used directly without having to store any data. The data provided is JSON-stat 2 format, a lightweight and straightforward JSON standard for data dissemination. JSON-stat supports several classes of responses, where possible values of classes are dataset, dimension, and collection. The dimension class includes information about the different attributes. Each attribute has a category node that defines the various options for that attribute. An example of a dimension can be sex. Which then will have the label sex, and the category consists of one array for index or Id and one object for labels where the key is the index, and the value is the label name.

So the Dimension sex would consist of the categories Males and Females as shown below.

```
[
  {
    "length": 0,
    "id": null,
    "class": "category",
    "index": 0,
    "label": "Males",
    "note": null,
    "unit": null,
    "coordinates": null
  },
  {
    "length": 0,
    "id": null,
    "class": "category",
    "index": 1,
    "label": "Females",
    "note": null,
    "unit": null,
    "coordinates": null
  }
]
```

Metadata and data are totally different in JSON-stat. Having a simpler structure is necessary for the data to be processed and displayed. Using dataSet.toTable(type: "arrobj") JSON-stat provides the data in a more readable structure that is an array of objects similar to a normal JSON file.

Also, while converting it checks whether a municipality is still active, and if not, it is given a validTo value. This is an example of how a dataset looks after conversion to a more readable structure.

```
[
 {
   "value": 5390,
   "ContentsCode": "Persons 16 years and older",
   "Tid": "1970",
   "Kjonn": "Males",
   "Nivaa": "Basic school level",
   "Region": "Halden",
   "RegionNumber": "0101",
   "gyldigTil": 2019
 },
 {
   "value": 4985,
   "ContentsCode": "Persons 16 years and older",
   "Tid": "1980",
   "Kjonn": "Males",
   "Nivaa": "Basic school level",
   "Region": "Halden",
   "RegionNumber": "0101",
   "gyldigTil": 2019
 }]
```

The Frost API

The Frost API can only retrieve data from one weather station per request. Since the application will be visualizing weather data across the country, multiple weather stations will be needed. Therefore, the data is stored in a database, eliminating the need for numerous requests each time the information is requested.

Frost offers a variety of data using JSON format. As the data sets to visualize are definite, not everything they offer is of interest. Therefore, only the relevant data will be stored in the database. That will be data of geometric values, attributes that provide weather stations(city, municipality, county), time, weather element, Id, and value. This will result in a database schema that looks like this:

6.1.2 Data Selection

As we proceed along the pipeline, the following step concerns specifying the subsets of the data to visualize[11]. Data sets must be classified, and a selection method must be devised to perform the visualization. Choice of selection depends on the desired application and is up to the developers to determine. As the objective of the development goes towards giving the users the ability to pick and choose the data set to plot, the application must provide an interface to do so.

Both SSB and frost API provide data sets that are possible to visualize geographically, but only the subsets containing position data can be used because that is the information needed to visualize geographically. However, this does not apply to all the data they provide. Instead, only those containing data that can be located will be used. Of what is included, the user will be able to choose from according to their own needs. Thus, the user will be able to browse through the available data and visualize it on maps. The data structure will then give a position according to Norway's arrangement. The mapping will then occur within Norway's territorial, the eleven administrative regions, counties, and the 365 municipalities (as of 2020). Therefore, the purpose is also achieved in that the visually enhanced data is also located.

SSB

Depending on the user, data can be displayed as a heatmap or choropleth. The default time is automatically set to the first time in the dataset, which can later be changed by the user displaying the new data. A user can also choose between displaying municipality or county, and if neither is selected, the type of region will be found through an algorithmic method.

Frost

The user can choose between all the different elements momentarily stored in the database and select a time.

6.1.3 Data to Visual

The general idea is to map data values into graphical entities or attributes. Occasionally, mappings must be preceded by processing such as scaling, shifting, filtering, interpolation, or data sub-sampling[11]. It is the values and attributes that define the visualization. For instance, an object's data is spatial; the visualization would be a geographic coordinate, giving the location-specific information.

SSB

In some cases, SSB data can contain values for regions that do not exist according to the administrative areas. This is because Norway has, throughout the years, been changing the divisions of municipalities and counties. This results in values for regions that do not exist on the map. Regardless of the division and the name, the location still exists. In order to connect data values from SSB to the updated administrative areas described in The Norwegian Mapping Authority, a match between value and area must be made. When the issue arises, the area name is checked to see if it exists; if not, it is checked to see where it has existed and is renamed accordingly. In addition, the value must correspond to the merged area. In order to measure this, the average is calculated using merged areas.

An example describing this occurrence is where three municipalities were combined into one. On January 1st, 2017, Sandefjord, Stokke, and Andebu were merged into one municipality, Sandefjord. Until 2017 SSB only provides data values separately for the three municipalities. Because the map does not show previous administrative areas, the value average of the three municipalities was calculated into one value for Sandefjord.

When all the data is prepared, and the necessary combinations have been done, connect the data provided from SSB to the respective match in the GeoJSON and update the JSON. A GeoJSON now contains data properties from the SSB. To visualize this on a map with choropleth, changes must be done to find a color representing the value.

Taking all the values and separating them across a quantile scale will give us all the data split up into intervals of similar sizes. We chose to do 10 percentage intervals that we connected to a unique color. Now all the values have been given a color representing the position in the quantile scale.

Frost

Frost offers approximately 1335 different weather stations in Norway. Some stations were specialized for a specific type of element. This means that when a user requests values for a particular year and element, it might only offer around 150-200 values in Norway. Now, this is not enough to cover the entire map. Therefore, interpolation is necessary to find the estimated value between two points. An estimated value will cover the area where the points are found using inverse distance weighting. Inverse distance weighting is based on the assumption that values that are closer to each other are more likely to correspond to their function. In order to save memory only 10000 different points are calculated across Norway. This gives an accurate enough estimated value for Norway where we don't have a station. This number can be increased but this will slow down the speed but also cost more as the hosting server will have to be upgraded.

$$z_p = \frac{\sum_{i=1}^{n} \left(\frac{z_i}{d_i^p} \right)}{\sum_{i=1}^{n} \left(\frac{1}{d_i^p} \right)}$$

Figure 6.2: Inverse distance weighting formula

6.1.4 Scene parameter setting

The following section is devoted to viewing transformations. Scene parameter settings give the user the ability to specify attributes for the visualization that is relatively independent of the data. A scene is a graphical user interface that consists of a collection of elements that the user can apply to the application by tapping them[11]. The scenes will be in the form of menus and pop-up dialogues to get input from the user. In this case, the scenes will only be available where the parameter settings are significant for visualization[9]. By adjusting these settings, the user can change the characteristics of the visualization, allowing them to select the most appropriate visualization for their own needs. For this application, it will then apply that the user could choose the color of the map, the plots, and the page.

Regarding scene parameter setting, the principles of user-centered design were practiced. The guides address prioritizing user experience for the development of the user interface.

6.1.5 Rendering of the visualization

The final stage of visualization involves mapping from geometry data to the image. Multiple visualizations can appear on the screen simultaneously, so there are multiple representation mappings and renderings[11]. This part of the pipeline becomes very dependent on the underlying graphics library to achieve a graphic visualization. An overriding purpose will be to create a graphical visualization that presents all the information and only the information. Expressiveness thus measures the concentration of data. Visualization is effective when it can be interpreted accurately and quickly and when it can be rendered cost-effectively. Effectiveness thus measures a specific cost of information perception.

Chapter 7

User-Centered Design

An essential aspect of the development is creating an application with a view toward usability, as users will use the application and retain knowledge from it. Usability refers to human factors in terms of how humans perceive any product. The user-centered design (UCD) evolved from HCI, human-computer interaction, studies the way people relate to computing products. Paying attention to UCD will help create an application that meets the needs of its users and ensure the application maintains good usability. Among the many focuses of UCD is the user experience (UX). It includes the user's physical and emotional reactions to the product and their experience with the product[10].

It is possible to measure how well an application achieves its intended goal through user-centered design by implementing and practicing its principles. Time spent with users is a necessary part of the development process, and therefore implementing it correctly, can save time[10]. Also, it's vital to understand that usability principles can be implemented without the expertise of a user experience expert. Since visualization is established, the application must add the fundamentals of creating a great user experience. This part of the report will address how the visualization application was completed with design to benefit users and the user experience. As part of the development, the following factors are explained and assessed[6].

7.1 User-experience Goals

When users approach an application, they expect a specific kind of experience. Thus, what type of experience will be given to the users and what type of experience users want when approaching an application must be considered[8]. One thing to keep in mind is that users will use the application to perform tasks. Therefore, it is important to design an interface that meets users' expectations. To achieve this, we must spend time listing user-experience goals to help create a consistent experience throughout the application.

In pursuance of achieving a great user experience with the application, several factors were considered. An essential element is loading time since the application deals with visualizing data that is self-selected by the user[6]. This implies whether users are willing to wait for the visualization to load if it means a more rich experience is provided. In addition, we must familiarize ourselves with how users will interact with the application in terms of how much of the application will require user input and how much can be done through keystrokes[8].This will also affect how enjoyable the application is, so it is important to determine how effective and consistent the application should be.

7.1.1 Loading-time

A fast loading time is essential since a few seconds can feel like forever since nothing can be done other than waiting. A standard concept for today's loading time is using a loading bar. These are often inaccurate, as measuring the progress can be challenging. In this project, there are two forms of loading displayed to the user. One is when the data is getting fetched, while the other is when the components are lazy loaded. Loading dots moving with a message on top are displayed when some loading or computation forcing a rerender of a component happens. The reason for loading can be mainly a heavy computation like interpolation is ongoing and delaying the render, or the fetch is waiting for a response from either our API or the SSB API. The message displayed depends on precisely this. Depending on the API, if something is being fetched, it will say "Fetching Data" or "Contacting SSB." The message " Loading site " will be displayed when waiting for React to show the component. This gives the user a clear overview that something is happening and the site has not crashed.

7.1.2 Interaction

One of the objectives was to engage users in the application. Interactive visualizations will allow users to explore data independently. A well-crafted interface and animated transitions will accomplish this. Interactive visualization will provide a better overview of the data and tools for navigating the details. An exemplary implementation will then succeed, and more about developing an interactive interface is explained in x.x.

7.2 Design Principles

Through design principles, we can achieve the user-experience goals we have set[10]. Studying and understanding design principles can prevent them from making mistakes. They will also provide guidance based on humans' understanding and interpretations of their surroundings. Having a good grasp of design principles and predictive models will help effectively critique the application.

7.2.1 Principle of Proximity

The principle of proximity states that objects closer together have a more substantial impact on the perception of relationships among them[10]. A highly improved user experience can then be achieved through organizing and grouping items based on their function. The principle is especially applicable to the design of the application layout. An organized design will be easier to perceive and puts less strain on the user to find elements. It is crucial not to miss this simple principle to ensure that the application layout makes sense for users, not just for developers. The goal is to help users understand how the application works and evaluate the options quickly.

Before implementing the principle, several factors must be set for assessment to get the best possible usage. Since the website should have several pages, the inspection must be considered for the best possible user experience. A standard implementation would be to have a navbar that presents all possible pages. The location of the navbar will also be critical for context. Since the goal was to make the website as simple as possible, considering that the user audience is broad, it was decided to use a simple navbar at the top of the page that will be static for all pages.

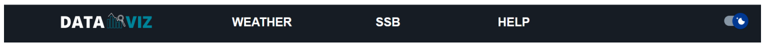


Figure 7.1: A consistent navbar throughout the entire application.

7.2.2 Principle of Visibility

The concept of attracting users' attention to an application's interface encompasses any element or action used to do so[10]. This applies to typeface, opacity, prominence, status, and colors. Application status can be implemented through this proposition. Visibility is substantial as it informs status while using the service. A case where this has been elemental is where a page that does not exist, for example, a data set does not exist, then status indication will be helpful. A button will then be displayed on the page to redirect the user to the homepage. The button will also follow the principles of visibility as the only option for users is to redirect to the homepage. The size, color, and location will be significant to achieve an explicit redirection.

Another condition where the visibility principle can be used is to provide visual feedback. This applies to situations where the application should respond to users' input as visual feedback indicates. The user's convenience was enhanced upon selecting the dataset to be visualized and then waiting for the data to be loaded. Feedback indicates that it has received information from the user, and interaction has occurred. Without this confirmation, the user is left confused about whether or not their action was received by the application.

7.2.3 Principle of Hierarchy

According to the hierarchy principle, the application must provide visual indicators to assist the user in perceiving how the application is organized[10]. A hierarchical organization aims to simplify the presentation of application features. Typical examples include drop-down menus and other navigational elements. The arranging should be set in a way that will be meaningful to users. When creating the application, we envisioned how all of its features would be organized. Since the application has a precise purpose, it does not consist of many pages and menus. The navigation between all possible pages is presented in a navbar, bestowed at the top throughout the application.

7.2.4 Principle of Models

Mental models refer to the use of models and figures in the application, and as developers, one must be well aware of them. Icons and language should accurately represent how applications function. The implementation can not happen arbitrarily because previous experiences apply knowledge to conceptualize how the new product may work. All icons and figures implemented were well thought out before use.

Figure 7.2: Raw data is converted into internal representation within the computer to then be display as a visual paradigm.

7.2.5 Principle of Afforance and Constraints

Affordance and constraints apply to develop actions that function in a way that makes it impossible to do the wrong thing[10]. This was an important principle to consider when the development of visualization was implemented. The goal of the application is to get the user to the visualization; thus, the path must be self-explanatory and precise. Accordingly, the application must be designed for the proper use and constrained from improper navigation. To achieve this in the best possible way, we observed users see what mistakes they could make to possibly limit options or anticipate their workflow.

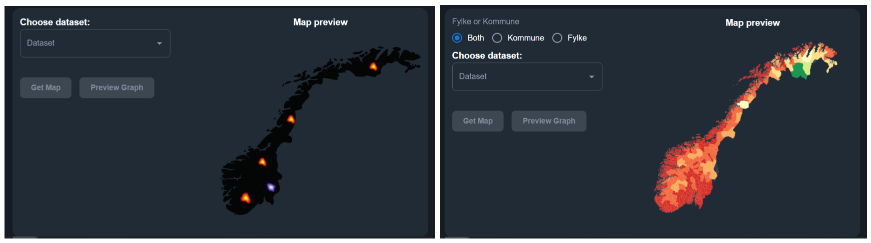


Figure 7.3: The figure refers to the choice of visualization tools, where other alternatives than what is possible are omitted.

7.2.6 Principle of Consistency

The principle maintains that users learn and understand applications more quickly when they are consistent with what they already know[10]. Consistency within the application is critical because it reduces users' cognitive burden and eases them into learning how it works. Consistency is applied to the visualization pages that are SSB and Weather. Since the two visualizations happen on different pages, the layout must be consistent, so the user is familiar with the interface regardless of which site they're on. Hence, the application will behave according to the user's expectations, and actions will have a proper outcome. This was achieved by keeping the entire website on one theme, color, font, and layout.

Chapter 8

The Prototypes

In order to test usability and obtain user experience, prototypes are built. Prototypes are not intended to be final products but rather are a step in the development process that will lead to testing and improvements. However, elements of the prototypes will be interactive enough to validate core concepts, even without the elements being fully functional. The first prototype creates a realistic visual concept of how the new product's functionality will look. In the second prototype, testing rounds and useful feedback are incorporated. This prototype makes it possible to see how an actual user interacts with the product and its functions to assess its final interactive elements and navigation.

8.1 Prototype 1

The user-experience prototype was made with an online collaboration software named Figma, a vector graphics editor with a prototyping tool. Figma is a free alternative to the popular prototyping tool Adobe XD, which is also a vector-based user experience tool developed and released by Adobe Inc. A prototype of a user experience models enough of a product's appearance and functionality so that the prototype can be used as a tool for user research. The tool allows detailed evaluation of visual elements and functionality in order to make changes. The process incorporates a user-centered design approach to define the usage context, design solution, and requirements needed.

8.1.1 Theming

The application will mainly visualize data from Norway. Knowing that the central theme for the web application was data, visualization, and Norway, a few design ideas came to mind. An example of a scrapped idea was going for a red, white, and blue color scheme with pictures of Fjords. However, the theme of choice was a color palette consisting of 6 colors representing the colors of the northern lights. Part of the application includes information about the weather, so a more of a nature theme makes sense. Both the northern lights and mountains are associated with Norway.

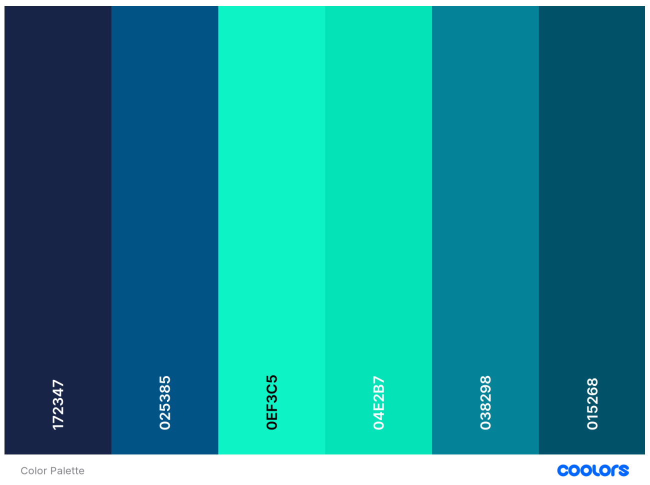


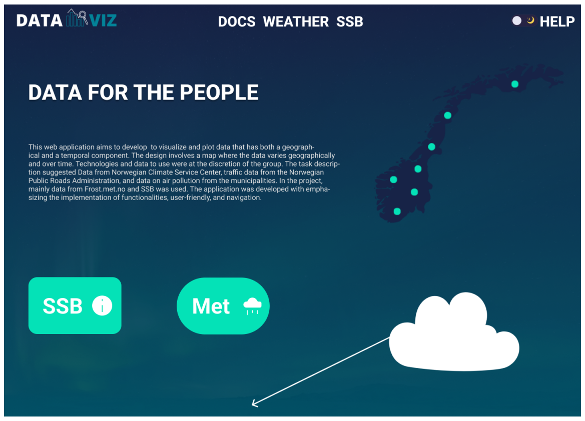
Figure 8.1: A display of the color palette made with coolors.co

8.1.2 Home Page

The main page of our web application is the home page. A user will arrive at the home page after entering the domain through a search engine. Everything is accessible from the home page and the home page works as a base for the website.

The home page consists of 3 sections. The first section displays a map of Norway and some text representing the website's objective. Although the map is merely a picture, it communicates to the user that this site deals with geographic visualisation of Norway. The lower part of the first section displays buttons that navigate to the SSB or weather page. A sky will move from the first section to the second section with the help of a spring-physics based animation library that handles all of our animations called React-Springs. The animation gives more life to the web page as a whole.

On the top of the first section is a transparent navigation bar. The navigation bar is fixed to the top of the screen and does not scroll down with the rest of the page. On the left is the logo, center contains links to the two main pages and on the far right is a night/light mode and a link to the help page.

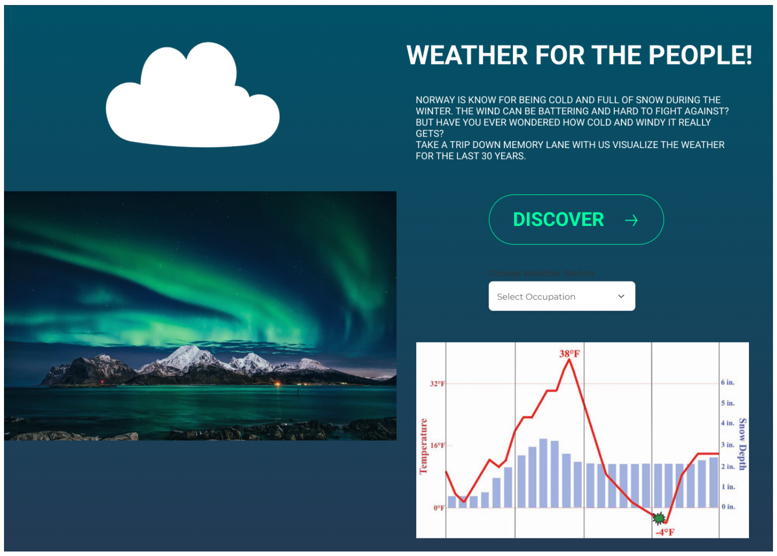


DATA FOR THE PEOPLE

This web application aims to develop to visualize and plot data that has both a geographical and a temporal component. The design involves a map where the data varies geographically and over time. Technologies and data to use were at the discretion of the group. The task description suggested Data from Norwegian Climate Service Center, traffic data from the Norwegian Public Roads Administration, and data on air pollution from the municipalities. In the project, mainly data from Frost.met.no and SSB was used. The application was developed with emphasizing the implementation of functionalities, user-friendly, and navigation.

Figure 8.2: Overview of the main page first section

The second section contains information about the weather page. This section is split into two vertically, where the left side has a visual feel while the right side is more informative. Clicking the "DISCOVER" button takes the user to the weather page. From the auto complete drop-down menu, user can select between all sources that offers temperature, and display a graph of the median temperatures per month for the last five years. This is a little demo of what happens on the weather page.

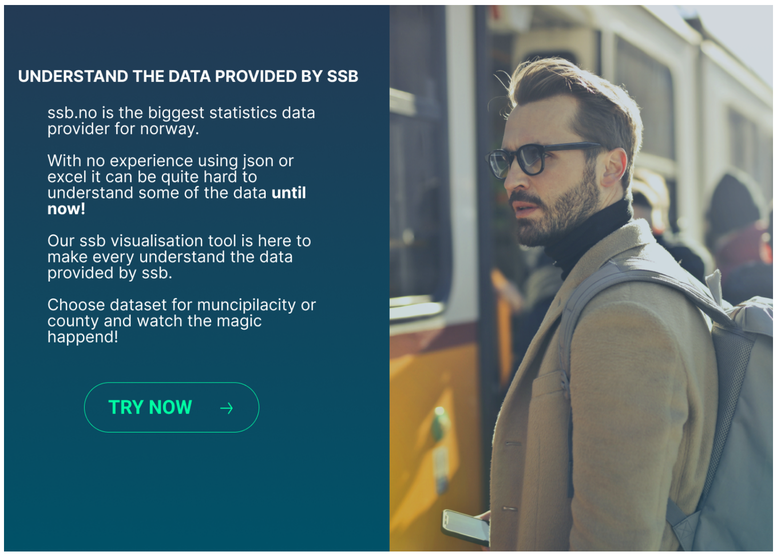


WEATHER FOR THE PEOPLE!

NORWAY IS KNOW FOR BEING COLD AND FULL OF SNOW DURING THE WINTER. THE WIND CAN BE BATTERING AND HARD TO FIGHT AGAINST? BUT HAVE YOU EVER WONDERED HOW COLD AND WINDY IT REALLY GETS?
TAKE A TRIP DOWN MEMORY LANE WITH US VISUALIZE THE WEATHER FOR THE LAST 30 YEARS.

Figure 8.3: Overview of the main page second section.

The last section has the same setup as the weather section just swapped. Visualization on the right and information about the SSB page in short on the left. SSB provides data mainly about households, incomes, and other areas of society.



UNDERSTAND THE DATA PROVIDED BY SSB

ssb.no is the biggest statistics data provider for norway.

With no experience using json or excel it can be quite hard to understand some of the data **until now!**

Our ssb visualisation tool is here to make every understand the data provided by ssb.

Choose dataset for muncipilacity or county and watch the magic happend!

Figure 8.4: Overview of the main page third section.

8.1.3 Help

The help page is mainly used as a guide for the user to follow in case needed. It helps the user understand how the application works by showing examples. The main focus for the help page was to keep it clean and simple. The help section also consists of several sections like the home page. Section top part of section one links to the help section for the weather or SSB site. Clicking the link will scroll the page down to the point where the user can find help about this. Bottom part of section one answers frequently asked questions, at the stage of creating prototype 1 it was hard to imagine what the perfect questions could be so therefor the figure does not include the questions. This sites also have a chat bot, the chat bot is not the most advanced chat bot to exists but it does the job fairly good.

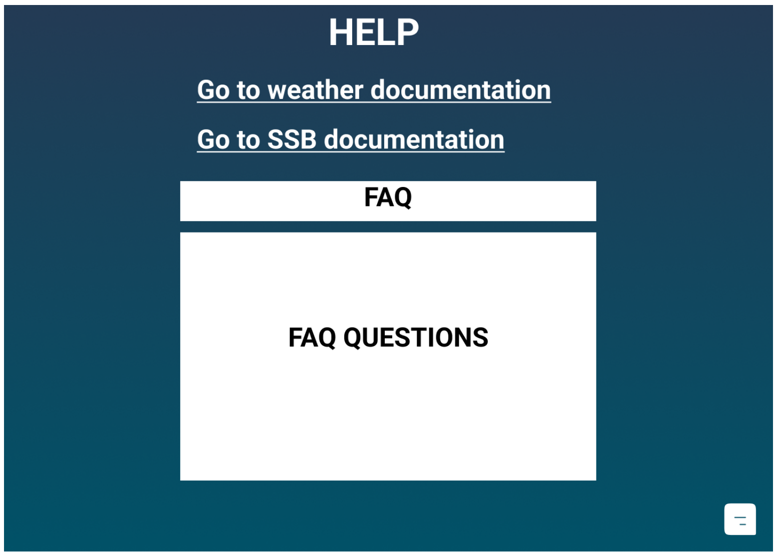


Figure 8.5: Overview of the help page.

By clicking on the "Go to weather help" in the first section the web page scrolls down to second section of the help page. The following is some information and a number of tips when using the weather application. There is also a button in second section linking the user back to the weather application.

8.1.4 SSB

As a result of pressing any of the SSB buttons the user will be presented with the SSB submit form page. SSB page consists of two different pages. The first page is the SSB user form page. All data sets from SSB require different sorting options so this is a necessary step. This is also the page where the user decides if it wants county or municipality and if the user wants heap map or choropleth. An PNG file example will be shown on the site regarding the choice of heat map and choropleth.

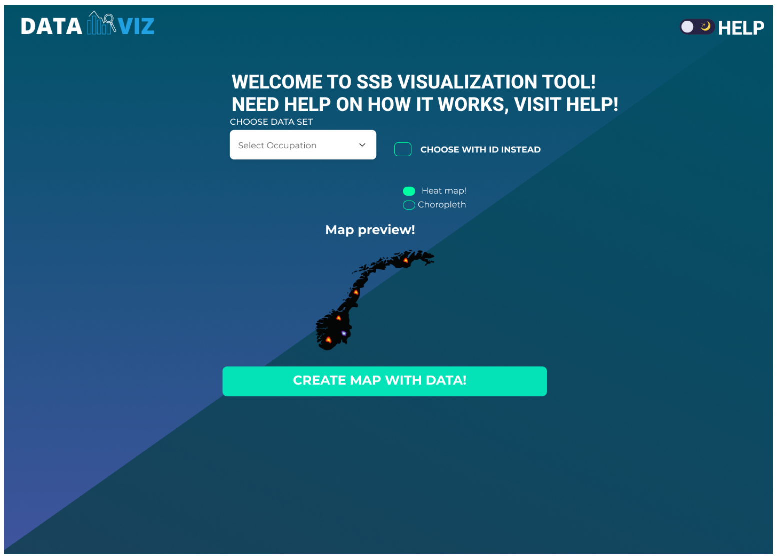


Figure 8.6: This is a sketch of how SSB submit page will look like.

The user than gets sent to loading screen showing some moving northern lights to entertain while the back ends works on preparing the data set.

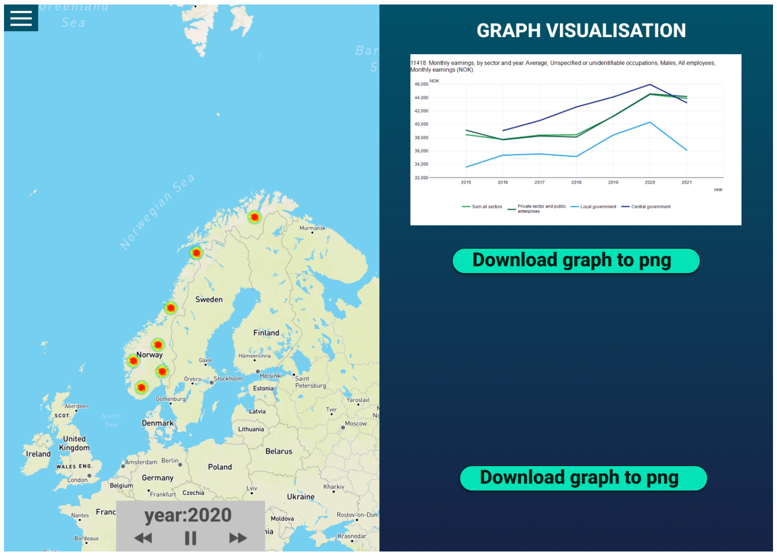


Figure 8.7: This is a sketch of how SSB map page will look like.

8.1.5 Weather

This page will show the temperature over the map. At the top of the page is a bar navigation with the logo on the left side and on the right side it has a Light/Dark mode switch with a help button. Below that, there is a slider where the user can select the time (in this case the date). Finally there is a map that will show the elements. The green box on the map is a choropleth map split up into 10000 different pieces where it will displayed a color representing the temperature. The numbers that are shown over the map are the number of stations that are in the certain area. These are clustered, but once the user zoom in more and more points will be displayed and the number will change.

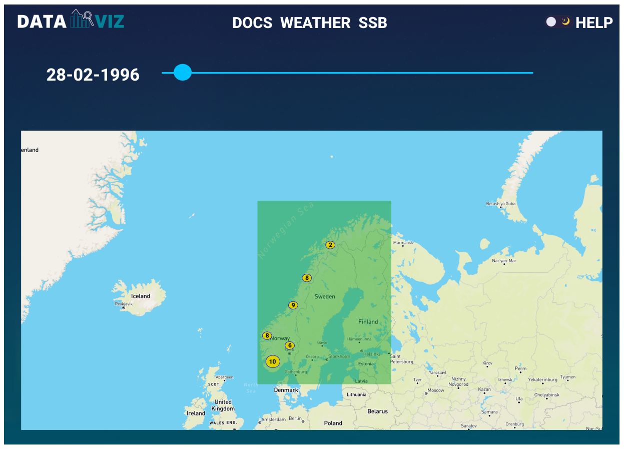


Figure 8.8: Overview of Weather page

8.2 Prototype 2

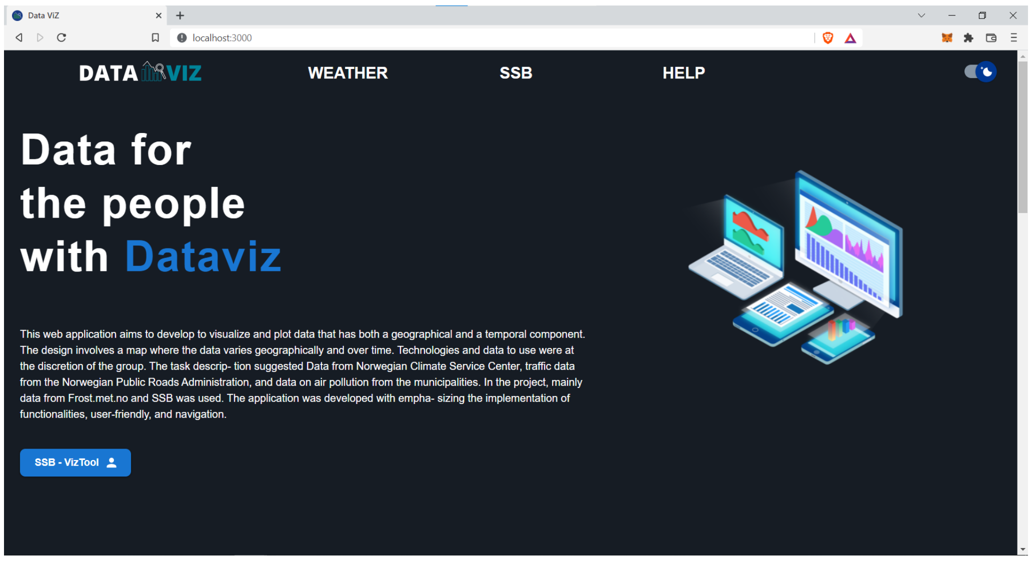
This prototype was created by using ReactJS, the front-end tool. The new interface design and its differences to Prototype 1 will be explained. Later on, the choices about technical aspects and architectural design will be analysed.

8.2.1 Theming

The theme has been changed from prototype 1. Using a northern light background theme made the site buttons and actions harder to present. Using a northern light theme made the site look more confusing and less clean in the sense as it took away the focus of buttons and switches. The goal from the start has been to make a simple site to navigate through but also use and focus on the visualization aspect. Material-UI Theme Creator was used to create the general theme but in short the primary color is #1976d2 and the secondary color is #0ef3c5. The theme starts as night but can be changed to light by hitting the switch. The use of the MUI function alpha has also been introduced helping reduce the transparency of colors. Other than that some customization was also done to the shadows of MUI as this something the theme creator offers.

8.2.2 Home

This image shows a redesigned version of prototype 1. The page still contains 3 sections. The first section displays different type of visualisation methods on different devices(phone, computer, tablet) instead of a map. This image displays the main objective of the site. It is also text representing the website's objective. The lower part of the first section display button that navigates to the SSB. There is no animation of Sky, compared to the previous prototype as this made the site more confusing. On the top of the first section is still a transparent navigation bar. In this prototype, the navigation bar is not fixed to the top of the screen and does scroll down with the rest of the page. On the left is the logo, center contains links to the three pages: Weather, SSB and Help and on the far right is only a night/light mode.

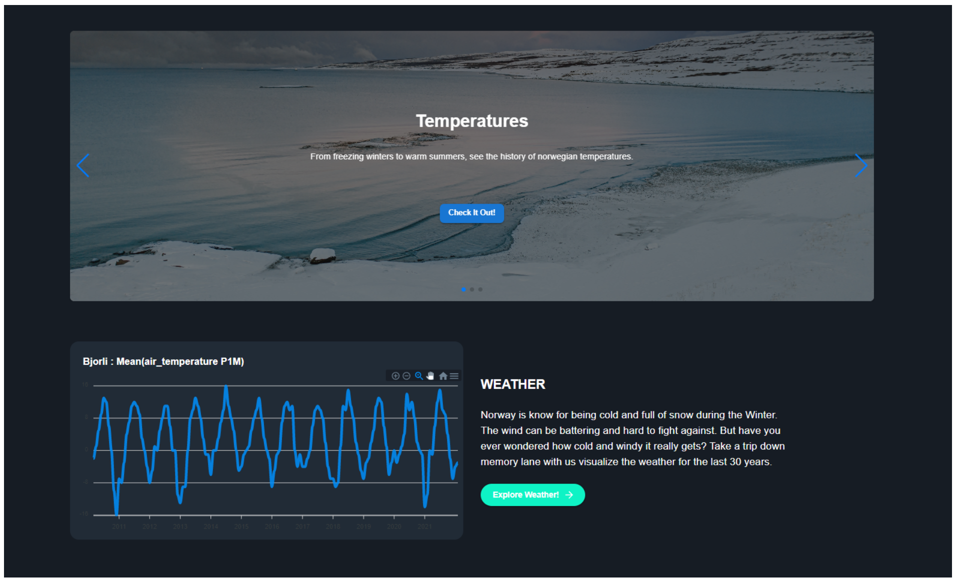


Data for
the people
with Dataviz

This web application aims to develop to visualize and plot data that has both a geographical and a temporal component. The design involves a map where the data varies geographically and over time. Technologies and data to use were at the discretion of the group. The task descrip- tion suggested Data from Norwegian Climate Service Center, traffic data from the Norwegian Public Roads Administration, and data on air pollution from the municipalities. In the project, mainly data from Frost.met.no and SSB was used. The application was developed with empha- sizing the implementation of functionalities, user-friendly, and navigation.

Figure 8.9: Overview of the main page first section

The first part of the second section consist of a slideshow. The slideshow represents different elements of weather. This will redirect the user to the weather page on click. The second part of the second section is a follows a well known website design setup that is picture left and informative text on the right. In this case the picture has been swapped out for a graph that can be interacted with. Fully downloadable and can also be zoomed in and out on. By clicking the "DISCOVER" button takes the user to the weather page. This section have undergo some changes sine prototype 1. Primarily the sky was removed which gave space for a slideshow. The slideshows adds a more innovative feel to it. The graph has also been moved to the left as it felt a little to crowded on prototype 1.



From freezing winters to warm summers, see the history of norwegian temperatures.

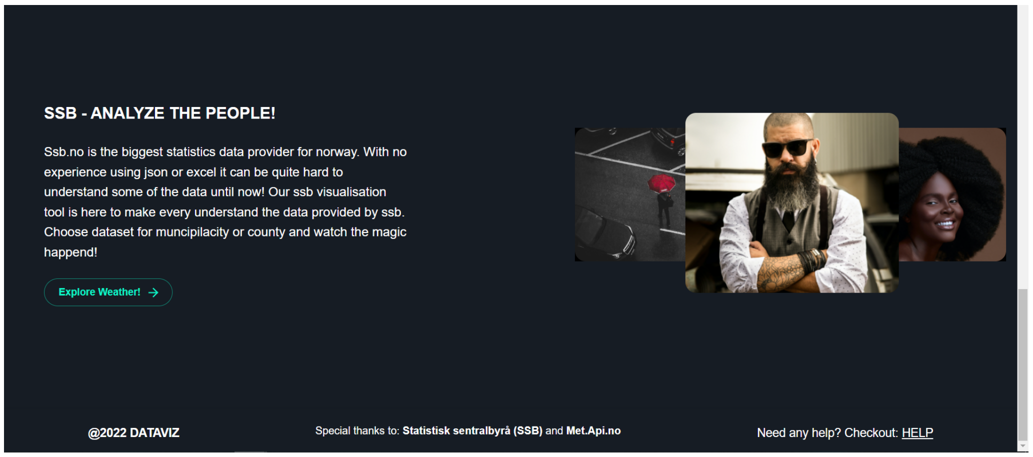
WEATHER

Norway is know for being cold and full of snow during the Winter. The wind can be battering and hard to fight against. But have you ever wondered how cold and windy it really gets? Take a trip down memory lane with us visualize the weather for the last 30 years.

Figure 8.10: Overview of the main page second section

Final section contains information about the SSB page. By splitting this vertically, on the right side is a gallery and the left side is an informative text with the button that takes it to the SSB page.

At the end of the page there is a footer where on the left side is a text "@DataViz", in the middle is a gratification for data providers and on the right side is a link to the help page. The home page feels much cleaner after removing the northern light theme.



SSB - ANALYZE THE PEOPLE!

Ssb.no is the biggest statistics data provider for norway. With no experience using json or excel it can be quite hard to understand some of the data until now! Our ssb visualisation tool is here to make every understand the data provided by ssb. Choose dataset for muncipilacity or county and watch the magic happend!

Explore Weather!
@2022 DATAVIZ
Special thanks to: Statistisk sentralbyrå (SSB) and Met.Api.no
Need any help? Checkout: HELP

Figure 8.11: Overview of the main page third section

8.2.3 Help

As the page was created it was discovered that the page turned out too long to only be one page. Instead of just creating a single page for Help, it was decided to create several parts, making the help page clean. The whole concept of help still remains the same: To help the user and provide a type of documentation for each part of the app. The visual still remains clean and simple, however, it suffered a few changes. At the top of the website is a navigation bar, where on the left side there is a logo, and on the right, there is a light/dark mode switcher with a button that goes back to the home page. The background has also been removed due to the changes in the general theme. Section after the navigation bar, part contains many parts. The first part contains two simple buttons: Guide and FAQ. Each button will visualize their respective components. These Buttons will be discussed in each chapter.

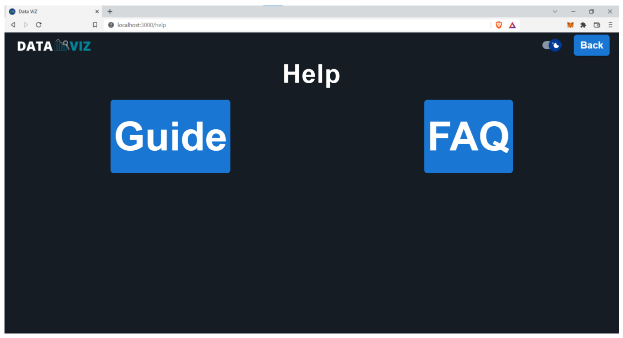


Figure 8.12: Overview of Help

By clicking on the Guide button, it shows a Guide for both SSB and Weather. The Guides are yet to be implemented as this would require some scripting and the group didn't find it necessary.

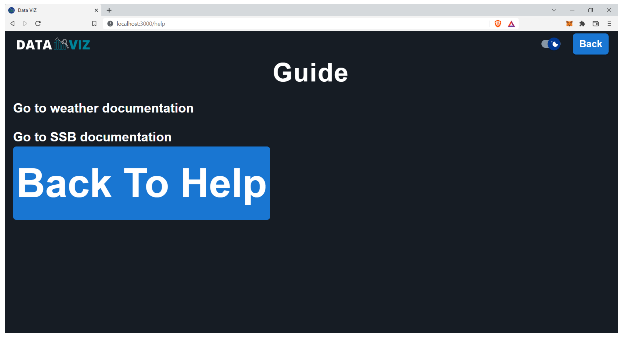
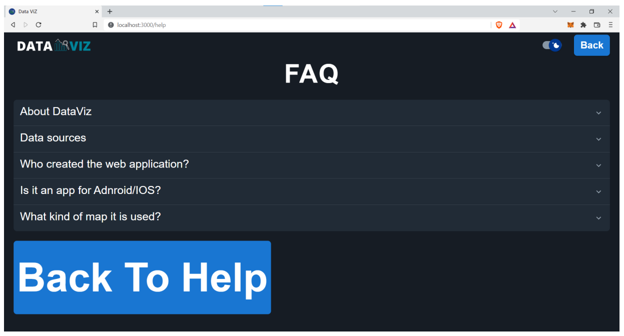


Figure 8.13: Overview of Guide

When clicked on the FAQ button, it shows the frequently asked questions. Although there were not too many questions that were asked, the group decided to select some of the questions that most probably asked. The questions are displayed in Accordion form, which is a type of menu that displays a list of headers stacked on top of one another. When the question is clicked, the answer will either be revealed or hidden. By using Accordion, it allows users to see what information is available on a page and decide quickly if the page is relevant to their needs. It also helps that the user does not have to scroll down in order to find the information that is needed.


DATA VIZ

Back

FAQ

About DataViz

Data sources

Who created the web application?

Is it an app for Adnroid/IOS?

What kind of map it is used?

Back To Help

Figure 8.14: Overview of FAQ

8.2.4 SSB

Ssb page has suffered the most significant changes from prototype 1 to prototype 2. The concept of having a submit page and a map page did not change, but there are few changes and some features that were added into it. This sub chapter will be divided in two parts: The submit page and map page.

SSB submit

The navigation bar consist logo that will redirect to the home page on click and a button that will navigate the user to the help page. The background has been affected due to the changes in the general theme. Under the navigation bar, it shows a text where it welcomes a user to the page. It also shows four boxes with four different colors and symbols: Work and income with blue color, Population with green color, Health with yellow color, and Education with red color. The numbers in the boxes represent the sum of data sets that SSB offer for each category. The main section of the site is the data selection part as described in chapter 6.1.2. It also has two radio buttons where it gives a user the possibility to choose between those two different kinds of map: heatmap and choropleth. When the Simple version of submit is selected, it has two radio buttons that select how the map will be divided: either by municipality or county. Below those buttons, there is a dropdown where it has a list of certain datasets. Under the dropdown, there are two buttons. The first button is "Get Map", the button that redirects data selected to the Mapview(which will be explained in chapter 8.2.4). The next one is called "Preview graph", this will change the image on the right into a graph displaying the some information about the dataset. On the right side of the simple version, there is a map that will be shown according to the user's choices. It is mainly there to display the difference between a heatmap and choropleth as the name might not be familiar to everyone.

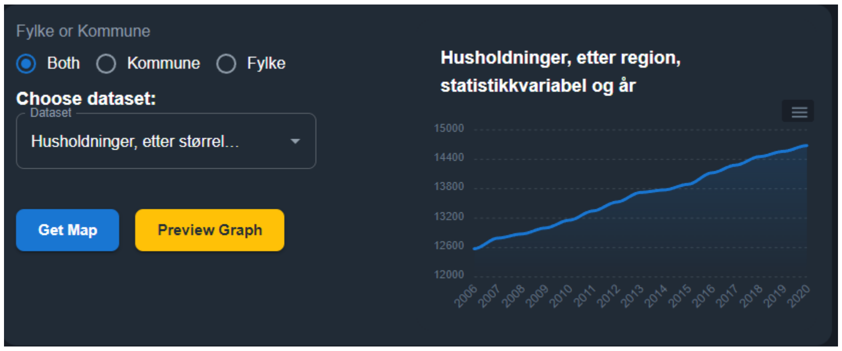
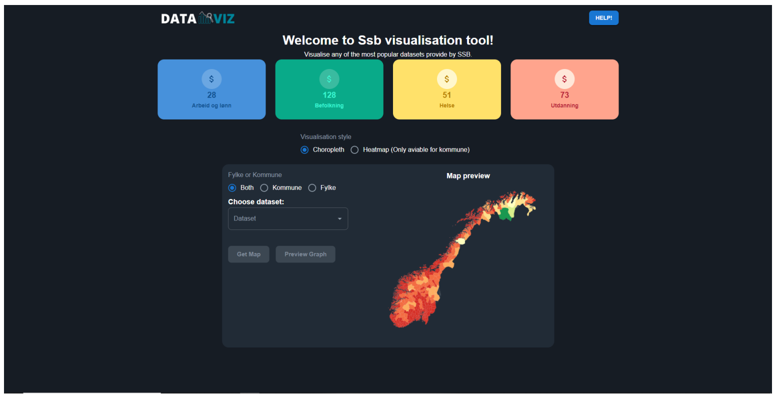


Figure 8.15: Display of how it looks when show graph is pressed

The concept still remains the same, however there are some details that were changed. The Id being removed is one of them, since the user when submitting the data, gets the url with id and it becomes possible to share with other people. Another change, but not drastic one, is the Choropleth has been changed to the default value when the user opens the ssb page. A feature that was added to this prototype was a preview of the graph can be accessed, as explained before. By fetching the different variables gave us access to the number of datasets, something we weren't aware of when we made prototype 1.



Welcome to Ssb visualisation tool!
Visualise any of the most popular datasets provide by SSB.

Visualisation style
[✓ Choropleth] Heatmap (Only aviable for kommune)

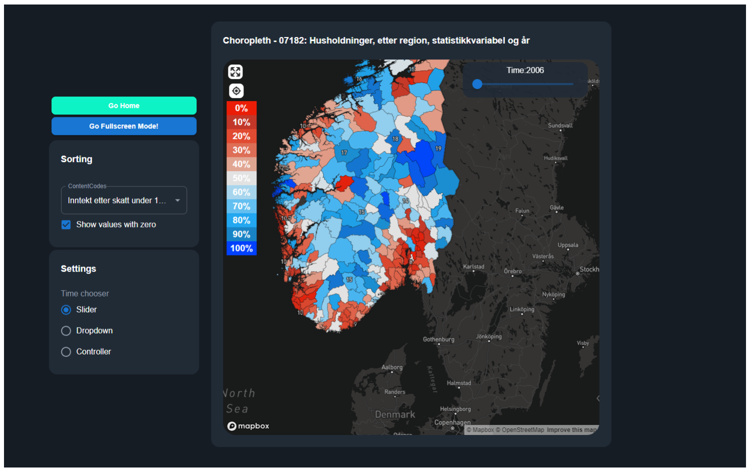
Fylke or Kommune
[✓ Both] Kommune Fylke

Choose dataset:
Dataset

Figure 8.16: Overview of Submit page

SSB Map

The concept of map does not run away from the prototype 1, but it is more redesigned. To start the map starts as full screen to match weather. However using the drawer on the left side the user can change to small screen mode. When using small screen mode the map is in the center of the page surrounded by other components. On the left side, there are three components. The first component is the button “Go fullscreen”. This button will make the map go full screen. The second component is sorting. This is where the data is sorted according to the criteria that is presented on the dropdown. The Settings is the final component. This component contains three radio buttons that will affect on the top right of the map. The first option is a slider, where the user can select the time by just sliding. The next option is a dropdown input version, where when it is open, then it shows a list of available times and lets the user select. The last version is the player or controller, where the time can change by itself when it’s on play. It is also possible to go back and forward on time. The user can in the settings option change the speed that the data will change.



Sorting

ContentCodes: Inntekt etter skatt under 1...

[✓] Show values with zero

Settings

Time chooser

[✓ Slider] Dropdown Controller

Figure 8.17: Overview of Map after data being submitted

When the map is in full screen mode, it shows the whole map itself. On the top left corner it is a menu, a drawer will come out displaying data selection options like sorting and settings. In the top right of the corner the time changer is displayed. When one of the regions is clicked, the screen is splitted vertically into two and the time changer is moved to the bottom of the screen. The map stays on the left side, while on the right it shows the two types of graphs: Area and Bar. On each graph it has a menu button on the right corner where it gives the user the ability to save the graph as SVG, PNG or CSV. If one more region is selected the graphs will change to show both the regions and the user can select a content type. Lastly the page also displays a list of all the colors and the value the color represent. Now some parts of the visualisation might be grey on the map. The reason for this is that SSB does not offer data for that region for that specific time mostly due to privacy reasons.

Although the concept of full screen on prototype 2 and the normal mapview on prototype 1 are quite similar, the prototype 2 gives the user more options on how to visualize data.

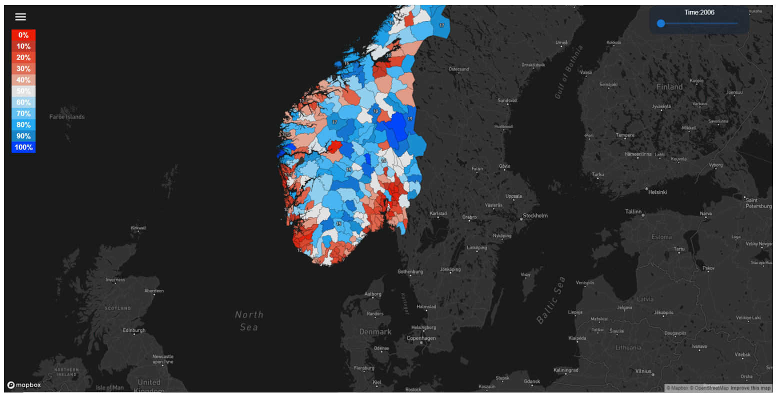


Figure 8.18: Overview of Map on full screen when graph is not visible

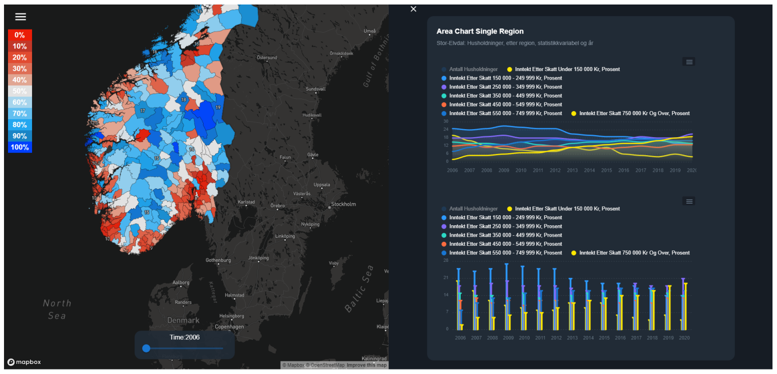


Figure 8.19: Overview of Map on full screen with graph visible

8.2.5 Weather

This page presents a full screen map. On the top left corner there is a menu button which is similar to the SSB. Bottom left corner there are three buttons and each of them change the elements on the map to either minimum, maximum or average temperature. On the bottom right, it shows the weather scale. The lower is the weather/temperature, the more purple is the color, while it becomes more red the higher is the temperature/weather. On the top right corner is the time controller, when you can get the temperature by selecting the month and year. Finally in the middle of the map, there is a map where it shows the color of the specific element. The map being full screen is one of main difference between this prototype and prototype 1. Another detail to notice is the time selector, this change happened because for a user it would be much clear to select time by using the drop-down input than just using slider.

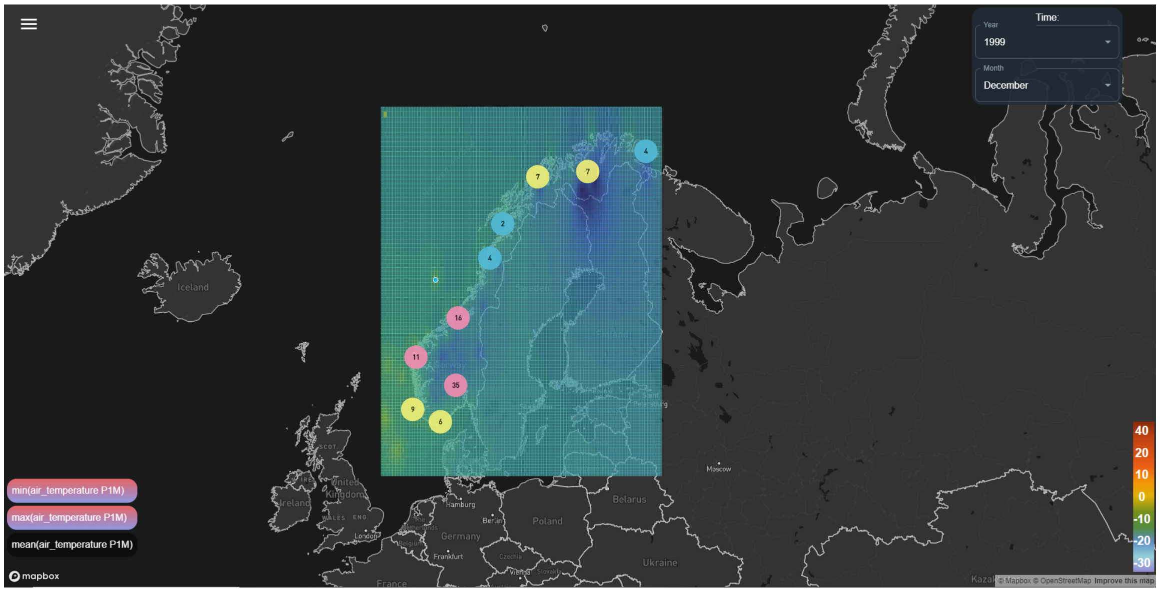


Figure 8.20: Overview of Map with weather mode

Chapter 9

Business Model

All the services used in this project are open-source or offers a free plan. By opting for a open-source services, it slows down the performance significantly and limits the potential that site can offer. In this chapter it will be explained the costs and potential profits if the best services were chosen.

9.1 Potential Cost Analysis

The default data only offers Norway because this is where data is accessible for free. Some sites offer historical weather data worldwide, but none of them are free, and they are all quite expensive.

One of the options that was considered was the openweathermap. This is one of the most used APIs for historical weather data and provides accurate weather data globally from 1980. Openweathermap historical data starts at the price of 7$ per location. Considering data from all cities with more than 100000 people, it would cost approximately 28.000$ and this is one of the more cheaper options. As a students, it is impossible to afford that value for an API, and therefore it was opted to go for an free API.

Mapbox offers several boundaries like municipality and counties. However this is comes at a cost. After contacting them and asking what something like this costs they responded saying the services starts at 20000$. The server started of by running on Heroku but have later been switched to AWS. AWS beats Heroku in price but is way more complicated to set up. The problem with Heroku is that it only offers PostGIS version 2.5 and our app rely on PostGIS version 3.2 or newer. AWS offers some pay as you go, which means you pay for what you use. AWS offers a free tier however it only offers 1 GIB of ram. The estimated price for AWS at a 16GIB memory usage for a site like this will be around $0.204 hourly but will increasing depending on how many users are on the site. This is at max 150$ a month but will only charge if a user is on the site.

These are mainly the potential Cost Analysis if before an official launch. There are many more if this were to be a real business. The website is not optimised for SEO so ads would have to be ran. Website crashes might happen so it is important to have someone how pays attention to server and fixes it in case something happens.

9.2 Potential Profit Analysis

The number one way to profit of a site like this is through sale or ads. It could be some complications running ads on a site like this without asking SSB or Met for permission since there data is free for use. According to absence with 50000 visitors a month will gain the website around 2000$ a year according to google absence calculator. This will barely cover the cost of AWS.

Another and more likely potential profit is selling the website or the server it offers. As earlier mentioned small media website might not have the money to effort a GIS specialist. They could rather use our service to display data for a specific year. It is near impossible to estimate the profits this can bring in as it is extremely unlikely and the website will need improvements before something like this can happen.

Chapter 10

Conclusion and Reflection

In this chapter the results of the application will be discussed as well as potential feature improvements.

10.1 Result

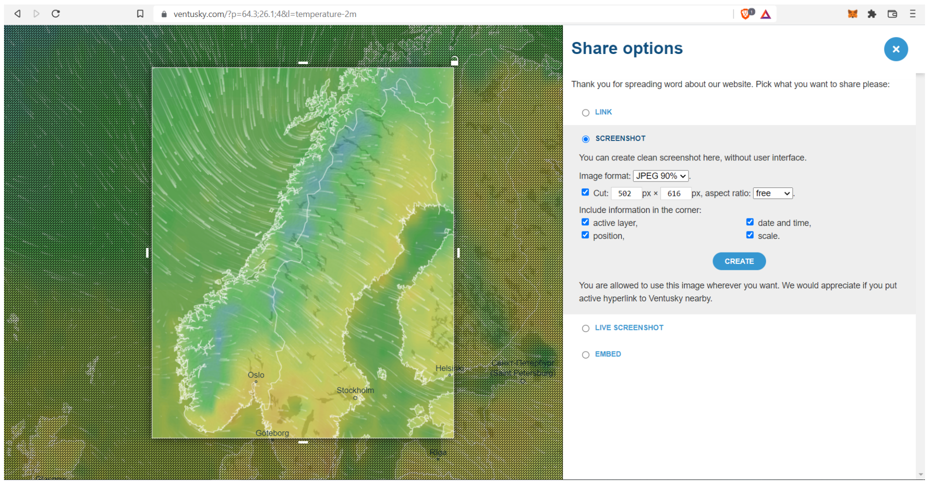
The application fulfills the main purpose of visualizing data that varies geographically and over time on a digital map. The data used comes from legitimate organizations and provides real data. Several algorithmic methods were used to match both area and values. This was then visualized on a map where the user can navigate and retrieve information. The application took into account both visualization principles and user-centered design principles. It was through these principles that the application gained its entirety. The completed application is only the beginning of what is possible. Lack of time was the biggest reason, but also lack of resources also had significance. Multiple sources were unwilling to share information, which could have helped us in numerous ways. Another problem was the little experience in GIS. Using GIS and understanding spatial data required more time and work than expected, which makes sense since there is a whole field of study in this area.

10.2 Features

Some features that can be implemented are as follows:

10.2.1 Capture screen of the map

This idea was taken from Ventusky website, where the user can capture the screen and save it. This would make a huge improvement when it comes to utility, as users could use those screen captures to either personal or business use.



Thank you for spreading word about our website. Pick what you want to share please:

SCREENSHOT

You can create clean screenshot here, without user interface.

Image format: JPEG 90%.

✓ Cut: 502 px × 616 px, aspect ratio: free.

Include information in the corner:

✓ active layer, ✓ date and time,
✓ position, ✓ scale.

You are allowed to use this image wherever you want. We would appreciate if you put active hyperlink to Ventusky nearby.

Figure 10.1: Ventusky feature of taking a screenshot over the map

10.2.2 Admin page

An admin page could be useful as it could let user request data and elements that they would like to have displayed. An admin would then log in and approve or disapprove of these requests. It was not a priority to create an admin page since it would not have a huge impact on the current web application. However, in the future when the application gets bigger, having someone who can get access to this data would be useful, for example, handling the database.

10.2.3 Global Expansion

It was always a possibility to have data that tracks various countries or continents. However, by not having enough sources to support this idea, it was considered to limit the range by having only Norway, since it is our residence. Also, SSB only provides data from Norway, which is one of the reasons why we were limited. In the future if there is a possibility to get better resources (economically, memory capacity and more), this could potentially expand to more countries and, eventually, continents.

Bibliography

[1] An overview of pgadmin - postegresql management tool. https://www.sqlshack.com/an-overview-of-pgadmin-postgresql-management-tool/.

[2] Postgis documentation. https://postgis.net/docs.

[3] Postgresql vs mysql. https://hevodata.com/learn/postgresql-vs-mysql/.

[4] Representational state transfer. https://en.wikipedia.org/wiki/Representational_state_transfer.

[5] Stack overflow survey. https://insights.stackoverflow.com/survey/2021.

[6] User centered design. https://www.interaction-design.org/literature/topics/user-centered-design.

[7] What is data visualization? definition, examples, and learning resources. https://www.tableau.com/learn/articles/data-visualization.

[8] Jesse James Garrett. *The Elements of User Experience: User-Centered Design for the Web and Beyond.* New Riders Press, 2 edition, 2010.

[9] Kieran Healy. *Data Visualization: A Practical Introduction.* Princeton University Press, 2019.

[10] Travis Lowdermilk. *User-Centered Design.* O'Reilly Media, 2013.

[11] Scott Murray. *Interactive Data Visualization for the Web.* O'Reilly, 1 edition, 2012.

[12] OnGres. Performance benchmark postgresql / mongodb. 2019.