PROGRAMMERING I GEOMATIKK, TBA4251

Kristina Jakobsen

https://github.com/Krijak/TBA4251

INNHOLDSLISTE

BAKGRUNN

Bakgrunnen for dette prosjektet er faget Programmering i geomatikk, TBA4251 på NTNU. Målet med faget er å kombinere kunnskaper innen informatikk med kunnskap om geografiske informasjonssystemer. Faget er selvlært, og gir svært god erfaring i å følge et produkt helt fra start til slutt, fra valg av teknologi, design og implementasjon til å ferdigstille et helt eget geografisk informasjonssystem.

Blant kravspesifikasjonene for denne oppgaven er det eksempelvis at GISet skal kunne behandle en begrenset mengde forberedte datasett for geografisk analyse. Applikasjonen skal være enkel å forstå, og skal kunne fungere som en attraktiv introduksjon til GIS.

Applikasjonens fokus og mål

Fokuset og målet for denne applikasjonen har vært å lage et enkelt og oversiktlig geografisk informasjonssystem med fokus på brukeropplevelsen (UX). Det skal være raskt å komme i gang, og enkelt å forstå. Fordi fokuset har vært på brukeropplevelse, er webapplikasjonen laget for alle skjermer, og er uavhengig av størrelse og touchegenskaper. Applikasjonen kan med andre ord ikke bare benyttes fra en PC, men også fra mobil og andre touchenheter (jeg har ikke hatt anledning til å teste applikasjonen på alle typer Apple-enheter, så jeg kan ikke garantere full kompatibilitet her).

TEKNOLOGI

Første steg i prosessen, er valg av teknologi. For dette prosjektet har jeg valgt å lage et vektorbasert GIS til web. Fordi jeg har valgt å lage en webapplikasjon, følger det spørsmål om hvilken teknologi som skal brukes for front-end, for back-end, webhotell (server) og en eventuell database.

Front-end

For front-end-delen av prosjektet, har jeg valgt å programmere i JavaScript. Valget sto mellom TypeScript og JavaScript. TypeScript er et supersett av JavaScript som kompilerer til JavaScript. Hva TypeScript tilbyr som JavaScript ikke gjør, er for eksempel en mer klassebasert objektorientert programmering (TypeScript, 2017). Disse egenskapene er spesielt gode å ha dersom prosjektet er av en viss størrelse. Fordi dette prosjektet er relativt lite, konkluderte jeg med at applikasjonen ikke kom til å bli omfattende nok til at det ville vært fordelaktig å bruke TypeScript i stedet for JavaScript.

En naturlig følge av å velge å lage en web-applikasjon, er å bruke HTML og CSS. HTML er et markeringsspråk (Wikipedia, 2016), og er det som definerer nettsidens elementer, mens CSS er det som definerer elementenes utseende.

Back-end

For back-end har jeg valgt å lage en nodeJS-applikasjon. Node er en server-side-plattform som bygger på Google Chrome sin JavaScript-motor. Node.js er plattformuavhengig, og fordi node.js er asynkront, fungerer det godt for applikasjoner med mye I/O (Tutorialspoint, u.d.).

Webhotell

For webhotell, har det stått mellom Amazon Web Services EC2 og Heroku. Det er en viktig forskjell på disse to tjenestene (Upguard, u.d.). Amazon Web Services tilbyr «Infrastructure as a Service» (IaaS), hvor de kun «leier ut» datakraft over nettet. Amazon Web Service er gratis å bruke i et helt år frem til en viss størrelse. Heroku, på den andre siden, tilbyr «Platform as a Service» (PaaS), hvilket er infrastrukturen pluss litt til. Heroku er også gratis å bruke frem til en viss størrelse.

For å bruke Amazon Web Service, må man selv installere programvare og drive vedlikehold. Platform as a Service er en litt bredere tjeneste, og man trenger ikke å gjøre ting som å drive vedlikehold. En bakside ved det er at man ikke får innsikt i den underliggende infrastrukturen. En direkte følge av at Heroku tilbyr en bredere tjeneste, er at det koster mer å skalere hos Heroku enn det gjør å skalere hos Amazon. Amazon Web Service sin gratis tjeneste tilbyr også høyere ytelse enn Heroku sin gjør.

Jeg konkluderte med at AWS er en tjeneste som passer bra for prosjekter av en viss størrelse og hvor det er ønskelig med høyere grad av kontroll over infrastrukturen. Heroku, på den andre siden, virker som et svært godt alternativ for nybegynnere som ønsker å komme raskt i gang. Nettopp for å komme raskt i gang, valgte jeg Heroku. Dette valget vil jeg komme tilbake til senere i oppgaven.

Et lærerikt alternativ til Amazon og Heroku ville vært å hoste selv på en server fra geomatikklesesalen, men det prosjektet er i skrivende stund ikke påstartet.

APPLIKASJONENS STRUKTUR

Biblioteker

For å lage denne web-applikasjonen, har jeg benyttet meg av en rekke biblioteker. To biblioteker for kartdelen av applikasjonen, og resten for et godt brukergrensesnitt og kommunikasjon med back-end.

Leaflet.js

Leaflet.js er et open source JavaScript-bibliotek for mobilvennlige og interaktive kart (Leaflet, 2017). Jeg har brukt leaflet til å tegne selve kartet, og for å tegne geoJSON-lagene på kartet.

Turf.js

Turf.js er et open source Javascript-bibliotek for beregninger på stedfestet data (Turf, 2017).

JQuery og JQueryUI

JQuery er et JavaScript-bibliotek for å gjøre det lettere å forholde seg til interaksjonen mellom JavaScript og HTML, samt å forholde seg til Ajax (jQuery, 2017). JQueryUI er blant annet brukt for drag-and-drop-funksjonaliteten for å endre lagrekkefølge, se «Hvordan bruke GISet».

Bootstrap-colorpicker

Et lite bibliotek for en pen colorpicker (Petre, 2017).

Front-end

Front-enden består av Main.js og Geometry.js. I Main.js er det meste av user interfacet definert, eksempelvis logikken bak hva som skal skje når bruker trykker på endre-lag-knappen (editLayer() og updateSidebarMenu()), hva som skjer når noen av lagegenskapene endres (layerChanges()), osv.

Geometry.js har flesteparten av funksjonene som handler om kartet og behandlingen av stedsfestet data, eksempelvis initialiseringen av kartet, og hva som skal skje når man trykker «merge layers». Ett lite unntak er bufferberegningene som blir gjort på back-end (se «utfordringer underveis»).

Back-end

Back-end består i all hovedsak av Index.js, og det er her serveren startes fra.

For flere detaljer og kommentert kode, se GitHub: https://github.com/Krijak/TBA4251

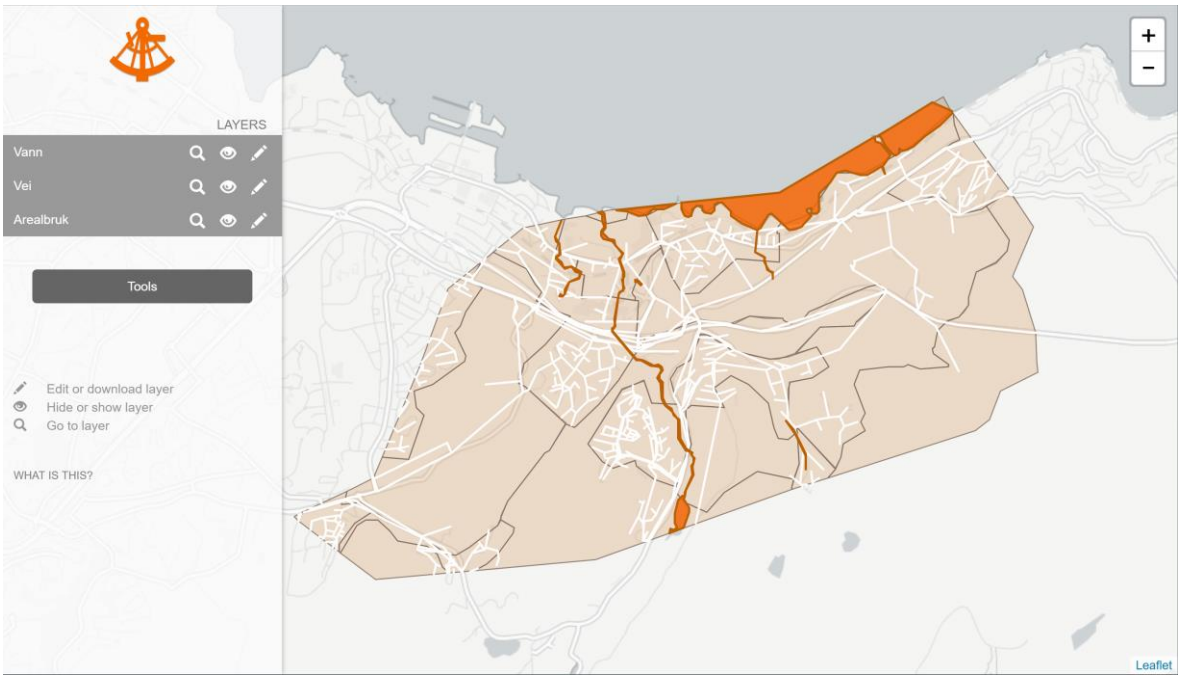
HVORDAN BRUKE GISET?

Følgende funksjoner er implementert:

- Endre farge, gjennomsiktighet og navn på lag
- Laste ned og slette lag
- Vis og skjul lag
- Panorer/zoom til lag
- Endre lagrekkefølge
- Romlige verktøy
    - Buffer
    - Union
    - Intersect
    - Difference

I tillegg fungerer alt på mobil.

Det første som møter brukeren er et kart med noen datalag, samt en sidebar, se figur 1.



Figur 1: Det første som møter brukeren

Bruker har følgende valg i sidebaren:

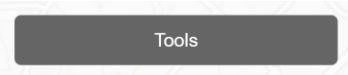
Rediger lag
Dersom man trykker på rediger lag, kommer rediger-lag-menyen til syne, se figur 2. Her kan man gjøre følgende:

- Endre navn på lag
- Endre farge og gjennomsiktighet for fyll (hvis polygon)
- Endre farge og gjennomsiktighet for strøk
- Slette lag
- Laste ned laget som geoJSON

Vis eller skjul lag

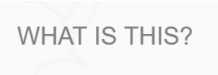
Panorer til lag

Endre lagrekkefølge
For å endre lagrekkefølgen på kartet, dras lagene i ønsket rekkefølge i layer-kontrollen, se figur 4.



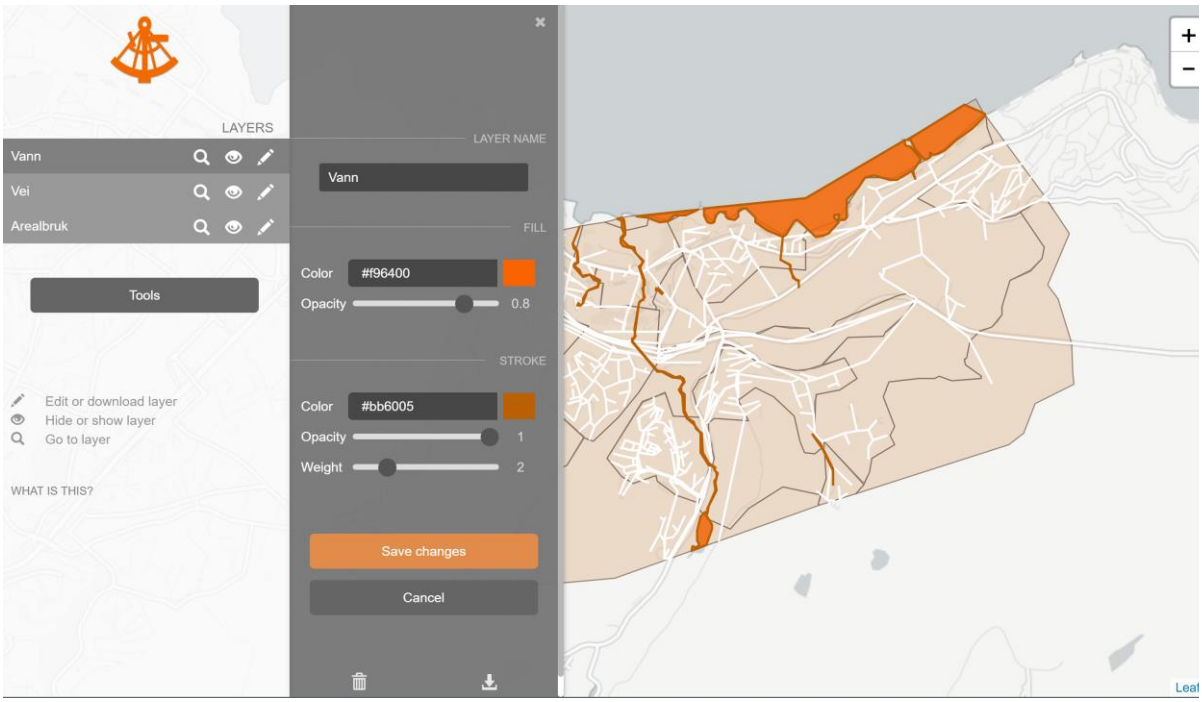
Romlige operasjoner
Dersom man trykker på «Tools»-knappen, kommer det en popup med en dropdown-meny, se figur 3. Her kan man velge mellom følgende operasjoner:

- Buffer
- Union
- Intersect
- Difference

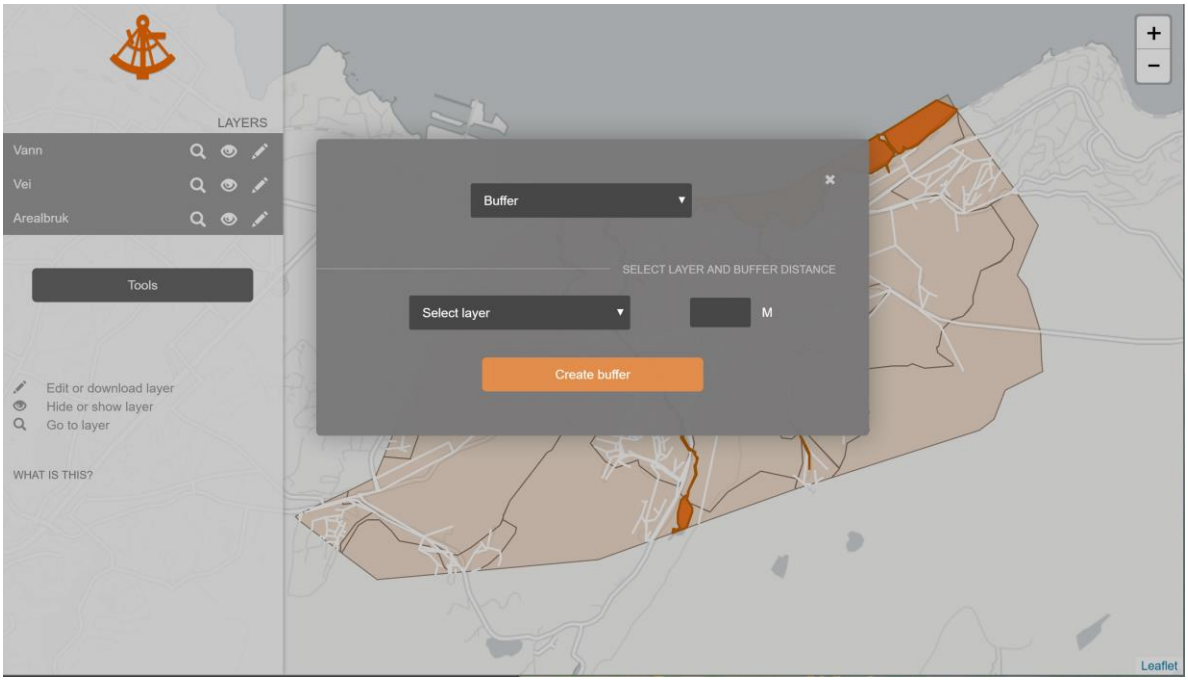


Hjelp
Består blant annet av use cases illustrert med gif.

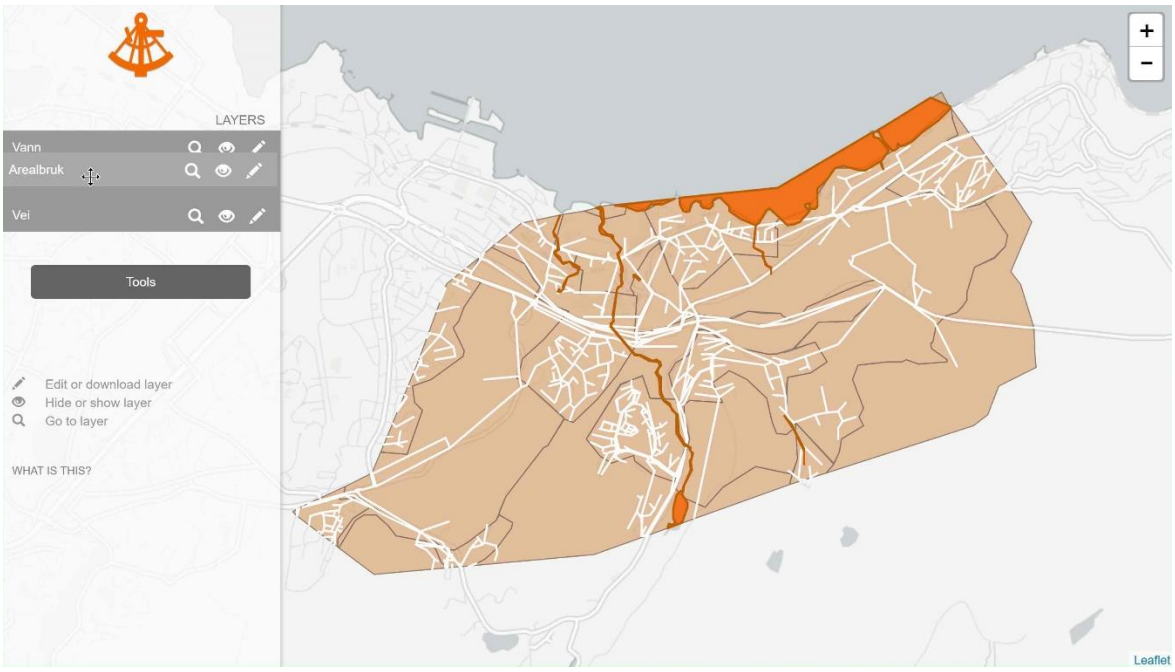
Dersom brukeren velger å utføre bufferberegning (den mest regnetunge operasjonen), får brukeren beskjed hvis laget er stort, og at utregningen da vil ta tid. Mens utregningen foregår, er det mulig å gjøre andre romlige operasjoner (ikke buffer), eller eventuelt redigere lag. Et laste-symbol dukker opp i sidebaren som viser at en bufferberegning pågår, se figur 5.



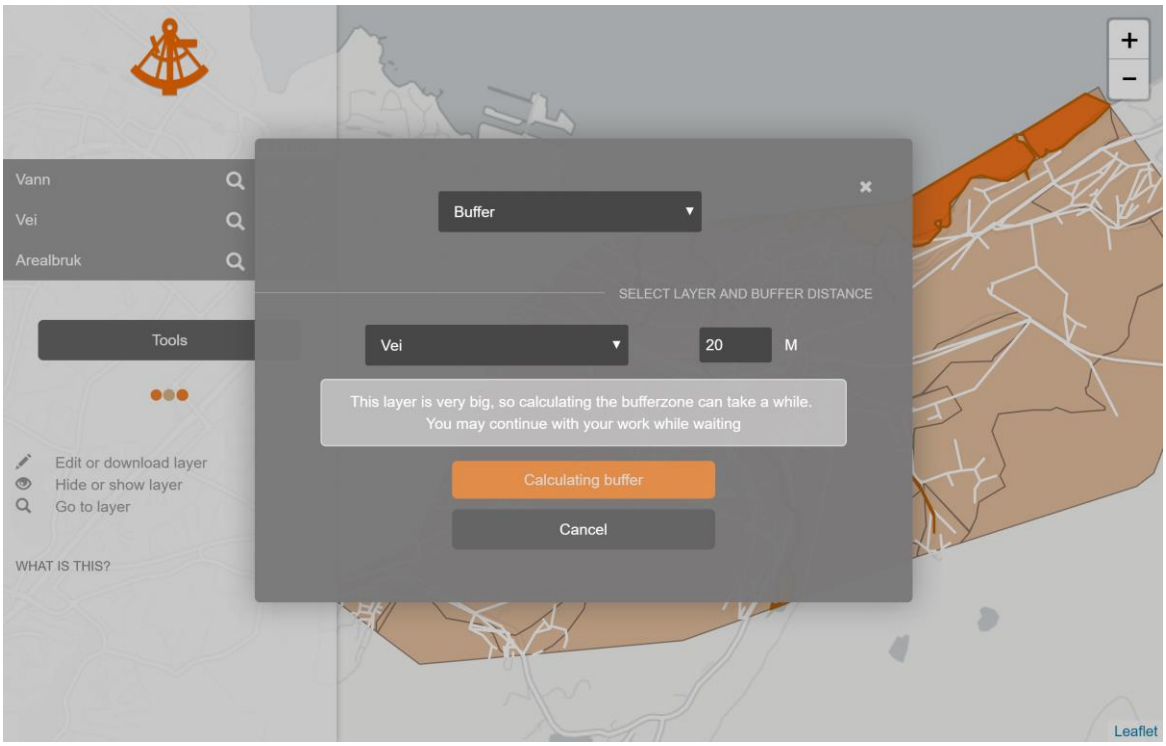
Figur 2: Menyen man får opp dersom man trykker på «Rediger lag»



Figur 3: Menyen man får opp når man trykker på «Tools»-knappen og valgt «buffer» i dropdown-menyen



Figur 4: For å endre lagrekkefølge i kartet, dras lagene i ønsket rekkefølge.

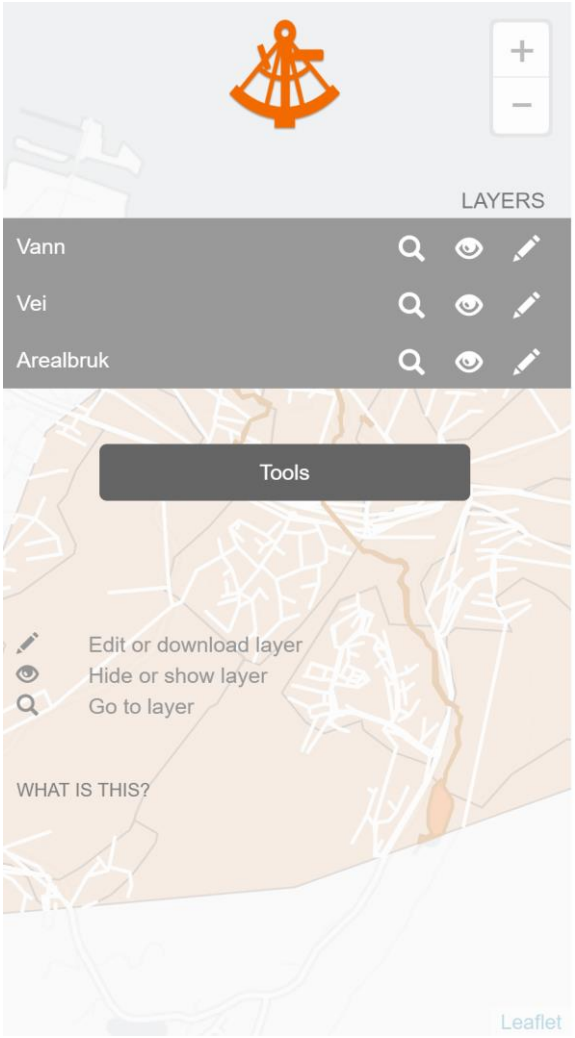


Figur 5: Dersom brukeren har valgt å gjøre en regnetung operasjon på ett av lagene får brukeren beskjed om at det kommer til å ta tid.

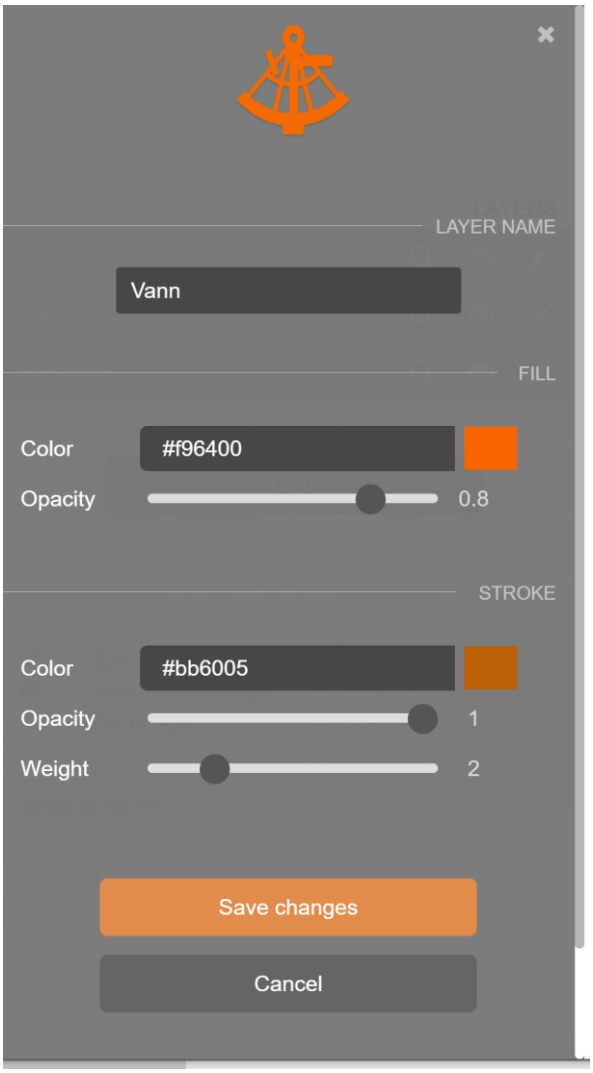
Mobil

De samme funksjonene gjelder også for mobil. Det som møter en mobilbruker, er kun sidebaren, se figur 6. Når man trykker på rediger-lag, fyller rediger-lag-menyen også hele vinduet, se figur 7. For å få se kartet, kan man trykke på logoen (sekstanten). Da forsvinner sidebaren og eventuelt også

menyen for å redigere lag, se figur 8. For å få sidebaren eller menyen for å redigere lag tilbake, trykker man kun på logoen igjen. Dersom man trykker på «Tools»-knappen, får man samme valg som man ville gjort dersom brukeren var på en PC, se figur 9.

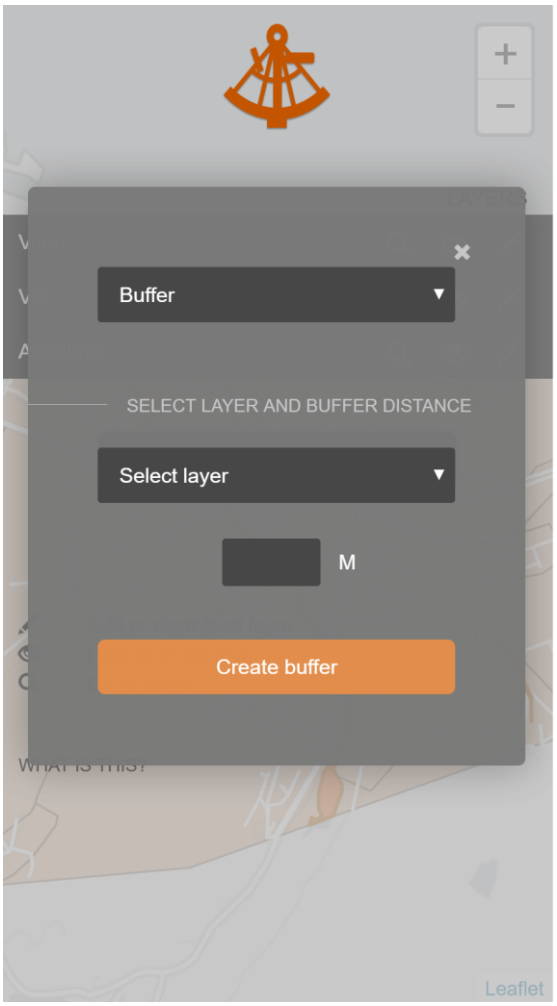


Figur 6: Vinduet som møter en mobilbruker



Figur 7: Når en mobilbruker trykker «Rediger lag»

Figur 8: For å få se kartet, kan man trykke på logoen



Figur 9: Når en mobilbruker har trykket på «Tools» og valgt «buffer» i dropdown-menyen

PROBLEMER UNDERVEIS

Noe av det jeg har vært mest usikker på i utførelsen av prosjektet, var om det var greit å utføre alle beregninger på client-side (front-end). Det er både fordeler og ulemper ved beregninger i front-end. Ett av lagene jeg brukte i utviklingen av applikasjonen, var et svært stort lag. Det viste seg å være ganske minnekrevende å utføre bufferberegninger på laget. Ved beregninger på client-side er det fordeler som rask responstid, men regneytelsen i dette tilfellet ble begrenset av nettleserens minne. Av og til ville bufferberegninger på det store laget føre til at nettleseren krasjet, og siden måtte laste på nytt. Dette var et stort problem all den tid applikasjonen ikke tilba funksjonalitet for å lagre progresjon. Det er flere løsninger på dette problemet, den enkleste ville vært å endre demolag til kun lette datalag. Fordi jeg på sikt ønsker å gjøre det mulig å laste opp egne datalag til applikasjonen, så jeg etter en mer langsiktig løsning. Jeg valgte å flytte bufferberegningene til server-side (back-end) og å bruke et rest-api. Det viste seg at Heroku sin gratistjeneste heller ikke tilba nok minne for å gjøre beregningene på server-side, men nettleseren krasjet i hvert fall ikke. En løsning på det problemet, kan være å bytte webhotell til Amazon eller til en som vi kan drifte selv. En annen fordel ved å ha den mest regnetunge operasjonen på server side, var at man kan fortsette med andre ting som å endre farge på lagene og å utføre andre beregninger enn buffer mens man venter på bufferberegningene.

```
MyApp.bufferAjax = $.ajax({
    url:"/api/buffer",
    type:"POST",
    data: theLayer,
    contentType:"application/json; charset=utf-8",
    dataType:"json",
    success: function(data){
            //print på kart
        },
        error: function(XMLHttpRequest, textStatus, errorThrown) {
            //send feilmelding
        }
});
```

Geometry.js (client-side)

```
app.post('/api/buffer', function(request,response){
    result = turf.buffer(request.body.layer, request.body.dist, 'kilometers');
    response.send(result);
});
```

Index.js (server-side)

Jeg ønsket helt fra starten av å fokusere på en god brukeropplevelse. Ett av problemene jeg har møtt i forbindelse med det, var at forhåndsvisning av lagfarge skulle skje med en gang brukeren endret farge i colorpickeren. Colorpickeren jeg har brukt, har en callback-funksjon som gir beskjed når

colorpicker-fargen endres. Colorpicker-callbacken ga beskjed også dersom man byttet fra ett lag til et annet uten å lukke «endre lag»-menyen, hvilket førte til at det gamle laget endret farge til det nyåpnede laget. Fra før av hadde jeg et system for å vite hvilket lag redigeringsmenyen tilhørte, men fordi JavaScript er et asynkront programmeringsspråk, ga callback-funksjonen beskjed før systemet oppdaterte hvilket lag som var aktivt. Jeg prøvde mye rart før jeg kom frem til en løsning som jeg syns fungerer veldig godt.

Bugs er alltid ett problem. Jeg har gjort så godt jeg kan for å luke vekk alle mulige bugs. En utfordring i seg selv kan være å finne bugsene. Som utvikleren av applikasjonen, er det ingen som vet bedre hvordan applikasjonen er ment å fungere, så det å trykke på ting i andre rekkefølger enn det som er meningen, fører fort til uønskede reaksjoner. Jeg har fokusert på å luke vekk alle bugsene for å optimalisere brukeropplevelsen. Et eksempel er å ikke oppdatere lagnavn dersom det nye lagnavnet kun inneholder mellomrom:

```
name != '' && !(/^ *$/.test(name))
```

LØSER APPLIKASJONEN OPPGAVEN PÅ EN GOD MÅTE?

Applikasjonen baserer seg på kravspesifikasjonene i oppgavedefinisjonen. Blant kravene har vi at GISet i hvert fall skal kunne behandle en begrenset mengde av behandlede datasett for analyse. På dette punktet oppfyller GISet kravet, men ikke noe mer. Grunnen er at det per nå ikke er mulig å laste opp egne datalag. Til tross for det kan applikasjonen behandle alle typer geoJSON-lag, også utover demolagene. Det er gjort slik at det skal være så uproblematisk som mulig å implementere funksjon for opplastning av egne data.

Et annet krav er at GISet skal tilby viktig GIS-funksjonalitet. Det finnes mange definisjoner av et GIS, så det finnes nok flere viktige GIS-funksjonaliteter som ikke er implementert. Et eksempel på det har jeg skrevet om i neste kapittel, «Veien videre».

To krav i oppgavebeskrivelsen baserer seg på design og brukergrensesnitt. Et av hovedfokusene til dette web-GISet har vært nettopp design og brukergrensesnitt. Jeg har prøvd å gjøre GISet så enkelt som mulig å bruke og å forstå. Det som møter brukeren er et kart med et sett med demolag og en sidebar. Sidebaren inneholder så lite som mulig, kun en liste med lag, en knapp for å få opp menyen for romlige operasjoner, samt en lenke man kan trykke på for å få hjelp. Jeg har valgt å ha de romlige operasjonene i en egen popup slik at sidebaren skal være så enkel som mulig, og ikke inneholde mange valg, hvilket fort kan bli uoversiktlig. Dersom jeg skulle laget et GIS for viderekomne, kunne det vært en ide å minimere antall trykk for å utføre en romlig operasjon (ved å eksempelvis ha hver operasjon i sidebaren), men jeg har vurdert det for denne oppgaven som viktigst å gjøre sidebaren enkel og oversiktlig.

Dagens mobiltelefoner begynner å ligne små PCer, og for å få en ekstra utfordring på brukergrensesnittdelen, har jeg gjort GISet responsivt og mobilvennlig.

Fordi definisjonen av et GIS er bred, er det mange mulige måter å løse denne oppgaven på. Dette prosjektet har fokusert på design og brukeropplevelse, og hvor godt den oppgaven er løst, er det nettopp opp til brukeren å bedømme.

VEIEN VIDERE

Det er fremdeles en rekke funksjoner jeg ville lagt til dersom det hadde vært mer tid. Den viktigste funksjonen ville vært å laste opp egne lag til en database (gjerne både shape- og geoJSON-filer).

En viktig GIS-funksjon som mangler, er det å kunne gjøre spørringer som baserer seg på informasjon lagret i lagene (lagegenskaper). En funksjon for å gjøre slike spørringer kunne vært interessant å implementere.

Jeg har valgt å droppe innloggingsfunksjon, fordi jeg mener at det ikke er viktig for noe som skal fungere som en introduksjon til et geografisk informasjonssystem. Det jeg derimot syns mangler, er en måte å lagre prosjektet på, slik at man kan jobbe vider med det på en annen enhet eller ved en senere anledning. En måte å gjøre det på uten å måtte lage bruker, kunne vært tilgang til prosjektet via URL (slik som ntnu.1024.no).

En viktig del av webdesign er å kunne lage design som er allment utformet. Et hovedpoeng her er at informasjon på en nettside ikke alene skal være kommunisert ved hjelp av farger. Det å for eksempel lage kart for fargeblinde kan fort bli en utfordring. Farger har sterke assosiative egenskaper. Det er med andre ord en egenskap som i stor grad er brukt av kartografer for å gjøre kart sebare. En måte å gjøre det lettere for fargeblinde å skille mellom ulike kartlag, kan være mønstre. En annen viktig ting, er å ha tilstrekkelig kontrast mellom fargene. I tillegg til funksjonalitet for mønstre, kunne det vært en ide hå ha med forhåndsdefinerte fargepalletter med farger som er lette å skille for fargeblinde og hvor fargene har gode kontrastforskjeller.

Ett av datalagene jeg benyttet meg av i utviklingen av GISet, var et relativt stort datalag. Bufferberegninger var eksempelvis svært minnekrevende for et lag av denne størrelsen, og gratisversjonen av webhotelltjenesten til Heroku tilba ikke nok minne for å utføre en slik utregning. En artig utfordring som kunne løst minneprobemet, ville vært å drifte en egen server fra lesesalen.

REFERANSER

jQuery, 2017. *jQuery.* [Online]
Available at: https://jquery.com/
[Accessed 2016].

jQuery, 2017. *jQueryUI.* [Online]
Available at: http://jqueryui.com/
[Accessed 2017].

Leaflet, 2017. *Leafet.js.* [Online]
Available at: http://leafletjs.com/
[Accessed 2016].

Petre, S., 2017. *Bootstrap Colorpicker 2.* [Online]
Available at: https://github.com/itsjavi/bootstrap-colorpicker
[Accessed 2017].

Turf, 2017. *Turf.js.* [Online]
Available at: http://turfjs.org/
[Accessed 2017].

Tutorialspoint, n.d. *Introduction to node.js.* [Online]
Available at: https://www.tutorialspoint.com/nodejs/nodejs_introduction.htm
[Accessed 2017].

TypeScript, 2017. *TypeScript.* [Online]
Available at: https://www.typescriptlang.org/
[Accessed 2016].

Upguard, n.d. *Heroku vs EC2.* [Online]
Available at: https://www.upguard.com/articles/heroku-ec2
[Accessed 2016].

Wikipedia, 2016. *HTML.* [Online]
Available at: https://no.wikipedia.org/wiki/HTML