Vtičnik v Grafani za napredno vizualizacijo vremenskih podatkov

Miha Lenko, Robert Meolic

Operato, energetske rešitve, d.o.o., Maribor, Slovenija
miha.lenko@operato.eu, robert.meolic@operato.eu

V članku obravnavamo problem vizualizacije vremenskih podatkov v sklopu tehnologije dinamično ocenjevanje prenosne zmogljivosti (DLR). To je obetavna komponenta zelenega prehoda energetike, ki omogoča bolj učinkovito integracijo obnovljivih in razpršenih virov energije samo z upoštevanjem vremenskih dejavnikov, ki vplivajo na segrevanje in hlajenje posameznih elementov. SUMO je modularna rešitev za DLR podjetja Operato, ki za vizualizacijo vremenskih podatkov uporablja orodje Grafana. V standardni različici Grafane ni namenskega pogleda na podatke o vetru, ravno veter pa ima največji vpliv na delovanje algoritma in si ga uporabniki zato želijo podrobno analizirati. Veter je za razliko od večine drugih vremenskih veličin vektor, torej ima hitrost in smer. Dodatno je smer vetra omejena z domeno vrednosti od 0 do 360 stopinj. V meteorologiji se za analizo podatkov o vetru pogosto uporablja prav poseben prikaz imenovan *rožа vetrov,* ki ga ni mogoče nadomestiti ali sestaviti z uporabo drugih grafikonov. Zato je bil izdelan nov vtičnik za Grafano. Za izvedbo naloge je bilo potrebno osnovno poznavanje jezika TypeScript, tehnologije ReactJS in formata SVG za opis vektorske grafike. Izdelana je bila kvalitetna aplikacija, ki omogoča sprotno osveževanje gradnikov in podpira uporabo različnih podatkovni virov. Vtičnik je že dosegljiv za namestitev iz uradnega kataloga.

Ključne besede:
Grafana
vremenski podatki
veter
roža vetrov
DTR
SUMO

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-286-772-0
DOI https://doi.org/10.18690/um.feri.8.2023.9

1 Uvod

Številni tehnološki procesi so odvisni od vremena. Na primer, vreme zelo vpliva na projekte v energetiki, izpostavimo lahko delovanje hidro, sončnih in vetrnih elektrarn. Za učinkovito vodenje takšnih procesov je potrebno natančno spremljanje sprotnih vremenskih podatkov in tudi vremenskih napovedi. Običajno se vremenski podatki zbirajo v lokalni bazi in se uporabljajo med produkcijo, pa tudi pri planiranju, optimiziranju, modeliranju in raznih analizah. Pogosto je potrebna vizualizacija.

Dinamično ocenjevanje prenosne zmogljivosti (DLR, angl. dynamic line rating oz. bolj splošno DTR, angl. dynamic thermal rating) je ena od naprednih tehnologij v prenosnih in distribucijskih električnih omrežjih, ki je zelo odvisna od vremenskih podatkov. Temelji na izračunu termične obremenitve daljnovodov in transformatorjev. Še posebej zanimiva je indirektna metoda, pri kateri dobimo rezultate brez namestitve dragih senzorjev, le z upoštevanjem vremenskih dejavnikov, ki vplivajo na segrevanje in hlajenje posameznih elementov. Slovenski operater prenosnega omrežja ELES je eden prvih TSO v Evropi, ki je indirektno metodo za dinamično ocenjevanje prenosnih zmogljivosti vpeljal v produkcijsko obratovanje. Njihova rešitev, ki so jo razvili skupaj s partnerji, se imenuje Sistem za ugotavljanje meja obratovanja (SUMO) [1]. SUMO je bil v zadnjem času s pomočjo podjetja Operato nameščen tudi v nekaj pilotnih projektih v tujini. Med novejšimi uporabami SUMO lahko izpostavimo še projekt TrafoFlex, katerega nosilec je bil SODO – Sistemski operater distribucijskega omrežja z električno energijo in ki se je ukvarjal z dinamičnim ocenjevanjem zmogljivosti distribucijskih transformatorjev [2].

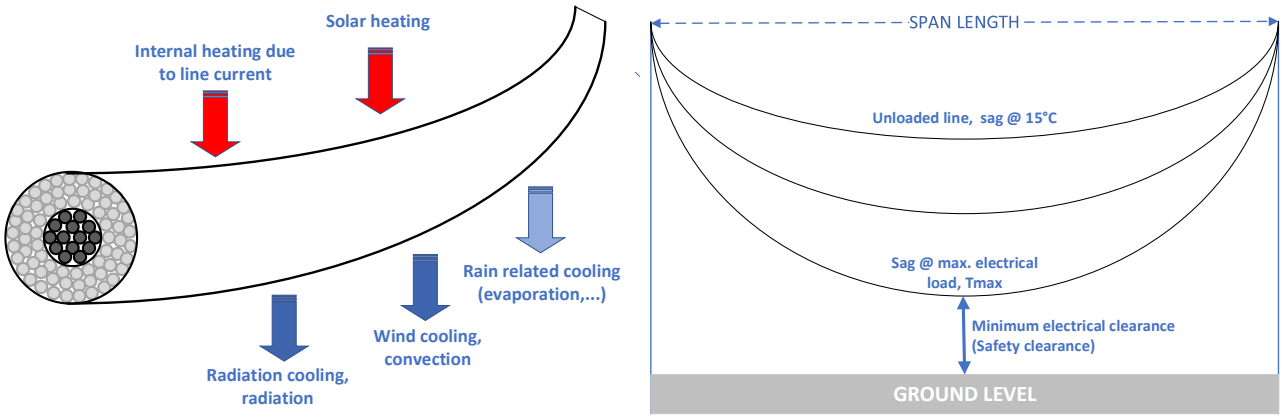
Polega izračuna DTR sta osnovni funkcionalnosti SUMO tudi vizualizacija ter analiza vhodnih podatkov in rezultatov. Analiziramo lahko tako zgodovinske podatke kot tudi napovedi za prihodnost. Trenutna različica SUMO 3 te funkcionalnosti doseže z uporabo odprtokodnega orodja Grafana (https://grafana.com/). Je zelo razširjen in dobro vzdrževan produkt. Na voljo je komercialna podpora. Najbolj primerna je za vizualizacijo časovnih serij, zmore pa prikazovati tudi tabelarične podatke. Podpira mnogo različnih vrst prikazov, ki jih lahko uporabniki dopolnijo s svojimi vtičniki.

V nadaljevanju članka je najprej podana motivacija za razvoj lastnega vtičnika v Grafani, šele nato sledi tehnični opis razvoja. Za izvedbo je bila uporabljena tehnologija ReactJS (https://react.dev/). Začetni del članka je nekoliko širši in nazorno pokaže, kako so kompleksne rešitve v informatiki odvisne od majhnih sestavnih delov kot je na primer vtičnik za vizualizacijo.

Opisan vtičnik prikazuje podatke o vetru in se uporablja v sklopu produkta SUMO. Ker pa je narejen zelo splošno, je uporaben tudi v številnih drugih projektih povezanih z vremenskimi podatki. Objavljen je v uradnem katalogu vtičnikov za Grafano.

2 Vremenski podatki v SUMO

Osnovna ideja tehnologije DTR je upoštevati vremenske razmere vzdolž vodnikov in ob transformatorjih tako, da omogočimo čim večji prenos energije ob še varnem obratovanju. Varnost obratovanja je pogojena s tem, da se zaradi segrevanja ne zgodi deformacija materiala, pri daljnovodu pa je pomembno tudi, da se ne poveča preveč njegov poves (slika 1). Uporaba DTR poveča varnost in pogosto tudi prepustnost prenosa energije v primerjavi s situacijo, ko za obratovanje uporabljamo statično mejo. Za vodnike je v Sloveniji statična meja izračunana ob predpostavki zunanje temperature 35 °C, hitrosti vetra 0,6 m/s s pravokotnim vpadnim kotom in sončnim obsevanjem 900 W/m². Statična oz. obratovalna meja pri transformatorjih je pogojena z njihovo zgradbo (npr. tip hladilnega olja) in izvedbo transformatorja (zidana izvedba, pločevinasta izvedba, izvedba v stavbi, jamborska izvedba, itd.). Ker pa so v večini časa vremenski pogoji ugodnejši od tistih upoštevanih za izračun statične meje, lahko ta dodatni pas izkoristimo za povečanje prenosne zmogljivosti, ob predpostavki seveda, da imamo dobre algoritme, zanesljiv informacijski sistem in natančne mikro-skalne vremenske podatke.

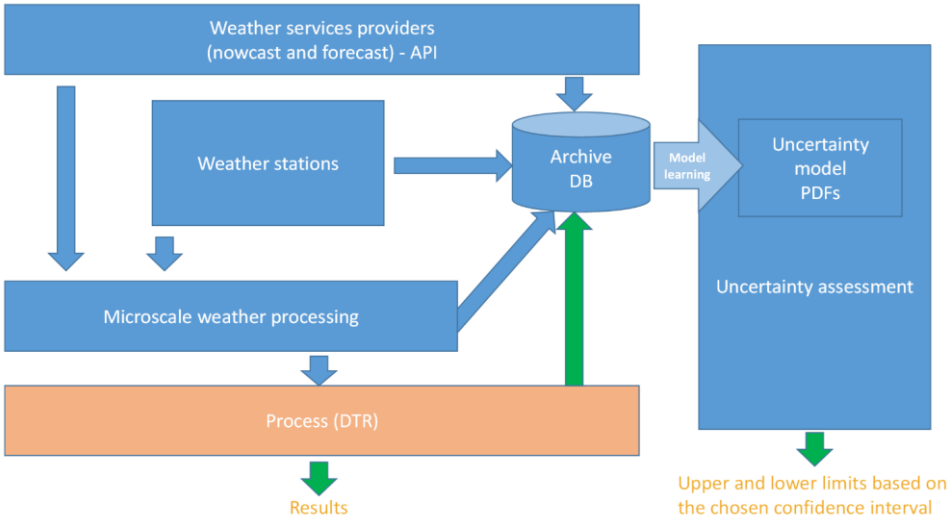


Slika 1: Vpliv vremenskih razmer na vodnike daljnovoda.

Vreme je skupek pojavov in stanj v ozračju, ki nas obkroža. V vsakodnevni rabi govorimo o vremenu tik nad površjem (tipično 2 m nad površjem), vendar pa so te vremenske razmere rezultat zelo dinamičnih in tudi kaotičnih procesov vzdolž celotne atmosfere do višine 10 km in še višje. Vremenske veličine je težko natančno meriti, kaj šele napovedovati. V sistemu SUMO potrebujemo mikro-skalne podatke (s prostorsko ločljivostjo okoli 200 m), ki jih pridobimo z izračunom na osnovi podatkov iz globalnih vremenskih modelov (tipična prostorska ločljivost 5 km) in meritev iz vremenskih postaj ob daljnovodih in transformatorjih (slika 2). V praksi imamo merjene podatke le za majhno število točk (vremenske postaje so velik strošek), vsi ostali podatki o trenutnem vremenu (angl. nowcast) so rezultat zgoraj omenjenega izračuna in zato vsebujejo napako. Napovedi vremena (angl. forecast) se izdelajo na osnovi podatkov o trenutnem stanju (ki vsebujejo napako) in omejeno zanesljivih rezultatov iz kompleksnih modelov, ki napovedujejo procese v celotni atmosferi v prihodnosti (zaradi zahtevnosti se taki izračuni tipično izvedejo le nekajkrat na dan).

SUMO uporablja algoritem DiTeR, ki so ga razvili na Inštitutu Jožef Stefan. Pri praktični uporabi se algoritem vsako minuto ločeno izvede za več tisoč lokacij. Za vsako lokacijo se z numeričnimi metodami pridobi rešitev nelinearnega sklopljenega sistema parcialnih diferencialnih enačb, ki izhajajo iz fizike termo-dinamskih sistemov. Problem ni rešljiv analitično. Uporaba matematičnega modela seveda vnese dodatne negotovosti glede rezultatov.

Ker se rezultati DTR uporabijo za upravljanje realne (in kritične!) infrastrukture elektro-energetskega sistema je pomembno, da se negotovost rezultatov ovrednoti. To izvedemo na osnovi zadostne množice zgodovinskih podatkov iz katerih izluščimo povezavo med vhodnimi podatki in napako rezultatov.

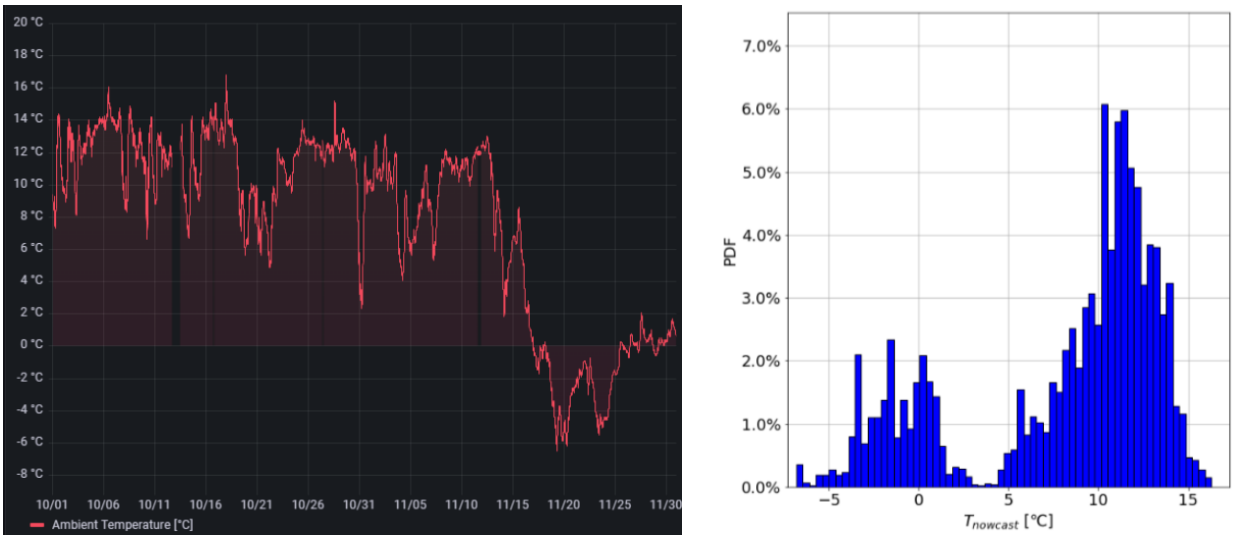


Slika 2: Uporaba vremenskih podatkov v SUMO.

Pri izvedbi DLR z indirektno metodo, kot je na primer SUMO, je za učinkovito vodenje sistema potrebno redno izvajati analize vhodnih vremenskih podatkov. Najpomembnejše primeri so:

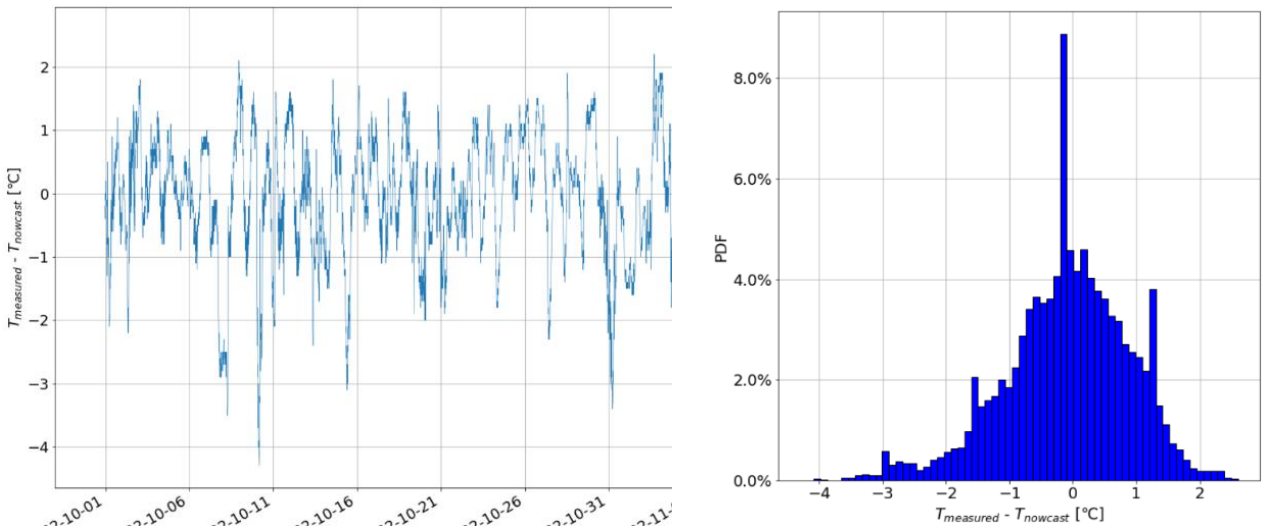
- s pomočjo analize vremenskih podatkov primerjamo različne vire (ponudnike) in se odločamo, kateri ponudnik bo najbolje zadostil potrebam izračuna DTR;
- analiza vremenskih podatkov pomaga določiti območja, za katera so podatki globalnih vremenskih modelov najmanj natančni in jih je zato smiselno opremiti z vremenskimi postajami;
- analiza vremenskih podatkov pomaga raziskovalcem pri oblikovanju boljšega modela za ocenjevanje negotovosti rezultatov.

Če se osredotočimo le na analizo nowcast podatkov (analiza napovedi ima še dodatno dimenzijo časa), sta najbolj osnovna pripomočka graf s časovnim potekom dogajanja in graf z distribucijo vrednosti (slika 3).



Slika 3: Vizualizacija vremenskih podatkov.

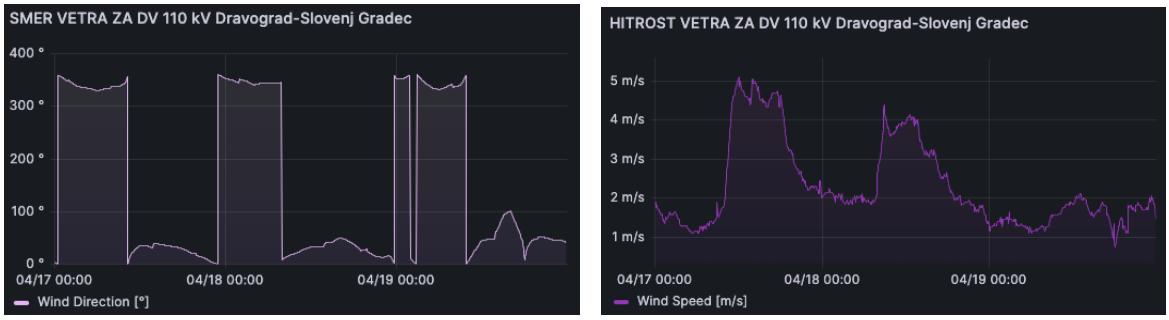
Če primerjamo dva vira podatkov, lahko uporabimo enak pristop, le da tokrat na grafih prikažemo časovni potek in distribucijo razlike obeh vrednosti (slika 4). Če je ena od vrednosti merjena oz. referenčna, potem lahko razliko vrednosti imenujemo kar napaka vhodnih oz. izhodnih podatkov.



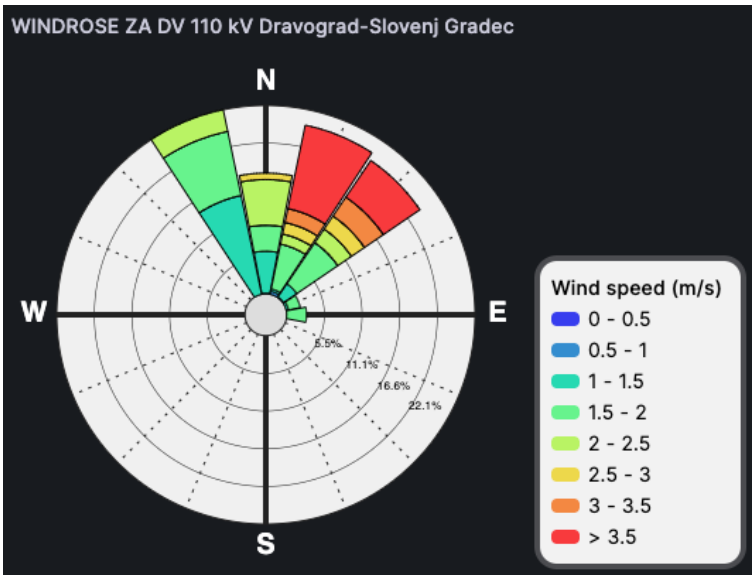
Slika 4: Primerjava dveh serij vremenskih podatkov.

3 Vizualizacija podatkov z rožo vetrov

Veter je za razliko od večine drugih vremenskih veličin vektor, torej ima hitrost in smer. Dodatno je smer vetra omejena z zalogo vrednosti od 0 do 360 stopinj. Ko vrednost za smer vetra naraste do 360 stopinj, preskoči na vrednost 0 stopinj, kar se na črtnih grafih izraža kot zelo moteča nezveznost (slika 5). Bolj pregledno je, če hitrost in smer vetra analiziramo na skupnem krožnem grafu (slika 6), ki se imenuje roža vetrov [3] (angl. Windrose [4]).



Slika 5: Prikaz hitrosti in smeri vetra z dvema ločenima črtnima grafoma.



Slika 6: Prikaz hitrosti in smeri vetra z rožo vetrov.

Kot ponazarja slika **Napaka! Vira sklicevanja ni bilo mogoče najti.**, je roža vetrov sestavljena iz več režnjev, v sak reženj pa iz več odsekov. V večini prikazov je število režnjev deljivo s številom štiri, ker s tem dobimo simetričen prikaz. Roža vetrov lahko interpretiramo kot dvo-dimenzionalni histogram, ki združuje obe komponenti vetra, hitrost in smer.

Roža vetrov najprej razdeli nabor podatkov po smeri vetra v prej omenjene režnje. Ti predstavljajo prvo dimenzijo dvodimenzionalnega histogram. Vsaka podmnožica podatkov znotraj intervala, ki ga določa reženj, pa se glede na hitrost vetra nadalje razdeli v odseke (glej legendo, ki je enaka za vse odseke). Vsak odsek (z izjemo najbolj zunanjega, ki predstavlja navzdol omejen interval) predstavlja navzdol in navzgor omejen interval hitrosti vetra, ter je na grafikonu obarvan z unikatno barvo. Vsak interval hitrosti vetra ima unikatno barvo in barva v vseh režnjih predstavlja isti interval. Odseki predstavljajo drugo dimenzijo dvodimenzionalnega histograma. Da pa lahko histogram interpretiramo, se frekvenca pojavitev podatkov znotraj posameznega intervala (odseka) ponazori z relativno velikostjo odsekov. Velikost odsekov je relativna tako, da največji reženj zasede celoten polmer grafikona. Na sliki **Napaka! Vira sklicevanja ni bilo mogoče najti.** je največji reženj tisti, ki je levo od oznake N (ki o značuje kot 0°), torej v interval med 326,25° ter 337,5° spada največ vrednosti. Nadaljnja interpretacija nam pove, da se znotraj tega intervala največkrat pojavljajo vrednosti med 1 in 1.5 m/s.

4 Grafana

Grafana je močno orodje za vizualizacijo podatkov, ki omogoča analizo in spremljanje različnih virov podatkov v realnem času. S svojim uporabniku prijaznim vmesnikom omogoča ustvarjanje privlačnih in interaktivnih prikazov, ki pripomorejo k lažjemu razumevanju kompleksnih podatkovnih nizov.
Glavni gradniki Grafane so podatkovni prikazi (angl. dashboards). Ti so sestavljeni iz posameznih grafikonov (angl. panels) ali drugih prikaznih gradnikov, kot sta na primer prikaz vrednosti ter prikaz seznamov. Vse omenjene gradnike lahko vidimo na sliki 7. Na vrhu podatkovnega prikaza imamo grafikon za izris časovnih vrst, kjer vidimo izrisane tri grafe (modra, rdeča in rumena črta). Pod njim je vodoravno razvrščenih pet prikazov vrednosti (*I [A]*, *ITH [A]*, …). Na dnu pa lahko vidimo (sicer ne cele) štiri sezname (INPUT DATA, DLR RESULTS, …).

Slika 7: Primer prikaza v Grafani.

Grafana podpira številne vire podatkov, vključno z bazami podatkov, oblačnimi storitvami, merilniki in raznimi orodji za spremljanje stanj sistemov, kot je na primer Prometheus. Omogoča tudi različne vrste vizualizacije, na primer črtne, tabelarne in geografske predstavitve, ter mnogo drugih vrst grafikonov.

Poleg vizualizacije podatkov Grafana ponuja tudi napredne funkcije, kot so opozorila in alarmi, ki uporabnike obveščajo o kritičnih dogodkih ali mejnih vrednostih. Integracija s sistemskimi orodji za sledenje napakam in spremljanje zmogljivosti omogoča hitro odkrivanje težav in odpravljanje napak. S svojimi sprožitelji spletnih dogodkov (angl. webhooks) omogoča integracijo obveščanja v druge aplikacije kot so na primer Slack, Discord ter Microsoft Teams. Slednjega smo pri Operato uporabili za opozarjanje ob izpadih merilnih postaj v sistemu SUMO.

Sodobna različica Grafane vključuje tehnologijo ReactJS. Gre za odprtokodno JavaScript knjižnico, ki se uporablja za gradnjo uporabniških vmesnikov. Njeno vzdrževanje koordinira podjetje Meta. Zasnovana je na konceptu komponent, kar pomeni, da lahko množico različnih gradnikov združujemo v kompleksne uporabniške vmesnike. Ravno ta lastnost tehnologije ReactJS omogoča enostavno izdelavo lastnih vtičnikov za Grafano, ker je potrebno samo zasnovati nov gradnik, ki ga Grafana potem samodejno integrira v svoj uporabniški vmesnik.

5 Izvedba vtičnika

5.1 Ideja in načrtovanje vtičnika

Ideja je bila preprosta, želimo grafikon v Grafani, ki na podlagi zbranih podatkov izriše diagram roža vetrov. Omogočati mora povezovanje vsaj na podatkovno bazo PostgreSQL, ki se uporabljamo v SUMO, vendar bi bilo priporočljivo, da podpira čim širši nabor podatkovnih baz. Želimo tudi da je vtičnik odziven ter se posodablja v realnem času brez potrebe po osveževanju strani. Grafana namreč omogoča nastavitev osveževanja prikaza v izbranem časovnem intervalu. Želeli smo tudi možnost prilagajanja prikaza ter spreminjanje barvne sheme, torej dve funkcionalnosti, ki smo ju najbolj pogrešali pri obstoječih rešitvah.

Prvi korak pri razvoju kakršne koli programske rešitve je načrtovanje. Z dobrim načrtovanjem se lahko izognemo mnogim težavam, ki nas čakajo v nadaljnjem procesu razvoja. Pa vendar je vse težave težko predvideti, zato moramo vedno najti dobro razmerje med časom, ki ga porabimo za načrtovanje, ter koristjo, ki nam jo načrtovanje vrača. Zaradi slabega poznavanja tehnologije ReactJS smo fazo načrtovanja skrajšali na minimum in se dokaj hitro podali v razvoj. Načrt za vtičnik je bil zastavljen na precej abstraktnem nivoju, ker smo se s tehnologijo ReactJS srečali prvič. Vedeli smo kakšen rezultat želimo, vendar zaradi nepoznavanja tehnologije nismo mogli natančno načrtovati specifik izvedbe vtičnika.

5.2 Pridobivanje podatkov

Na začetku je bila ena glavnih skrbi podpora za različne tipe podatkovnih baz. A smo hitro ugotovili, kako nas modularna zasnova Grafane elegantno reši teh skrbi. V ozadju vseh funkcionalnosti Grafane so vtičniki, ki se delijo v tri kategorije. *Aplikacijske razširitve* nas tokrat niso zanimali, ker pokrivajo razširitve same platforme. Potem pa so tu še *grafikoni*, te vrste je tudi naš vtičnik roža vetrov in *podatkovni viri*. Jasno razlikovanje med tema dvema kategorijama vtičnikov pomeni, da Grafana strogo ločeno obravnava pridobivanje in prikazovanje podatkov.

Podatkovni viri so vtičniki zasnovani za povezovanje na podatkovne vire, naj bodo to podatkovne baze (PostgreSQL, MySQL, MongoDB, ...), posrednike sporočil (Apache Kafka), tabele (Google Sheets) ali celo git repozitoriji (Gitlab in Github). Ti vtičniki se zaganjajo na strežniškem delu platforme ter glede na podane zahteve uporabnika iz podatkovnega vira pridobijo podatke. Te podatke nato v posebni podatkovni strukturi, imenovani *Dataframe*, pošlje odjemalcu, ki jih izriše ali kako drugače prikaže z uporabo grafikonov. Tak pristop zagotovi varnost, ker se odjemalcu nikoli ne pošiljajo poverilnice potrebne za dostop do podatkov v podatkovni bazi.

5.3 Orodja in razvojno okolje

Za razvoj smo uporabili Visual Studio Code, ki je s svojim širokim naborom razširitev priljubljeno orodje za razvoj spletnih tehnologij. Potrebovali smo tudi razvojno kopijo Grafane, ki jo podobno kot v produkciji, postavimo z uporabo tehnologije Docker. Ta tehnologija je enostavna za uporabo, razlikovati moramo le med dvema osnovnima pojmoma: slika in vsebnik. Slika Docker (angl. Docker image) je posnetek stanja zelo okrnjenega navideznega računalnika z minimalno namestitvijo programov in konfiguracij potrebnih za zagon aplikacije. Vsebnik Docker (angl. Docker container) pa je dejansko izvedena in izolirana instanca, ki temelji na sliki Docker. Vsebnik Docker si lahko torej predstavljamo kot minimalističen navidezni računalnik. Razlika med vsebnikom Docker in pravim navideznim računalnikom je v njuni arhitekturi, saj si, za razliko od prave virtualizacije, vsebnik Docker deli jedro ter nekatere sistemske komponente z gostiteljevim operacijskim sistemom. Je pa zagon in upravljanje vsebnikov Docker zato dosti lažje in manj obremenjujejo gostiteljski sistem.

V ukazni lupini prenesemo željeno sliko z ukazom:

```
> docker pull grafana/grafana:latest
```

Po končanem prenosu pa zaženemo lokalno instanco Grafane z ukazom:

```
> docker run -d -p 8080:3000 -e "GF_DEFAULT_APP_MODE=development" -v
"/user/operato/development/windrose:/var/lib/grafana/plugins" grafana
```

S tem smo zagnali vsebnik Docker z vsemi potrebnimi knjižnicami in izvedljivimi datotekami potrebnimi za izvajanje Grafane, do katere lahko sedaj dostopamo na naslovu http://localhost:8080. Izbrali smo vrata 8080, vendar bi lahko izbrali poljubna vrata, le navesti jih je potrebno v ukazu z zastavico -p <VRATA>:3000. S tem preusmerimo ves promet iz vrat 8080 na gostitelju na vrata 3000 znotraj vsebnika. Vrata 3000 niso poljubna, ker Grafana privzeto posluša na vratih 3000.

Z zastavico -v smo mapo /user/operato/development/windrose na gostiteljskemu sistemu vpeli v datotečni sistem vsebnika na /var/lib/grafana/plugins. Tako bomo lahko vtičnik razvijali na gostiteljskem računalniku, obenem pa bo dostopen Grafani znotraj vsebnika.

Zastavica -e nastavi okoljsko spremenljivko `GF_DEFAULT_APP_MODE` na vrednost `development`. To sporoči Grafani, da gre za razvojno različico ter s tem omogočimo nalaganje vtičnikov, ki niso podpisani s strani razvijalca.

Preden začnemo razvijati lastne funkcionalnosti vtičnika moramo postaviti ogrodje, s pomočjo katerega bo Grafana zaznala, da gre za vtičnik in ga naložila v svoj seznam vtičnikov. V ukazni lupini smo se premaknili v mapo /user/operato/development/windrose, ki smo jo vpeli v vsebnik Docker. Tam smo izvedli sledečo serijo ukazov:

```
> npx @grafana/create-plugin@latest
> yarn install
> yarn dev
```

Prvi ukaz je ustvaril ogrodje našega novega vtičnika in od nas zahteval nekaj osnovnih podatkov o vtičniku kot so ime, organizacija, kratek opis ter tip vtičnika. Ponudil nam je tudi možnost, da nam postavi Github CI, vendar smo to možnost zavrnili, ker ne uporabljamo platforme Github. Nato z ukazom `yarn install` namestimo odvisnosti, z ukazom `yarn dev` pa zaženemo skripto, ki spremlja izvorno kodo ter ob vsaki spremembi ponovno izdela vtičnik. Po izvedbi ukazov, ki lahko, odvisno od hitrosti internetne povezave, trajajo tudi nekaj minut, je potrebno vsebnik ponovno zagnati, ker Grafana išče nove vtičnike le ob zagonu.

Na tej točki se v Grafani na seznamu vtičnikov pojavi naš nov vtičnik z imenom, ki smo ga podali ukazu za izgradnjo ogrodja vtičnika. Ime lahko pozneje, poleg mnogih drugih lastnosti vtičnika, spremenimo v datoteki `src/plugin.json`.

5.4 Razvoj vtičnika

Sedaj se prične postopek izdelave vtičnika, zato se premaknemo v podmapo `./src`, v kateri se mora nahajati vsa izvorna koda vtičnika. Grafana uporablja programski jezik TypeScript, ki je razširitev JavaScript, sintaksi obeh pa se med seboj nekoliko razlikujeta. Vtičnik bo zgrajen po principu modulov. Vsaka datoteka, ki predstavlja svoj modul, mora vsebovati vsaj eno vrstico `export`. Takšen modul lahko potem uvozimo kjerkoli drugje v programu.

Ukaz za izdelavo ogrodja je že pripravil osnutek izvorne kode. Vstopna točka je modul, ki ga predstavlja datoteka `module.ts`. in v kateri je funkcija `plugin`, ki vrne vtičnik kot gradnik ReactJS. Na vrhu te datoteke se najprej uvozi nekaj modulov, med drugimi tudi SimplePanel iz datoteke `components/SimplePanel.tsx` (pri uvozu lahko končnice datotek izpuščamo, zato bomo v kodi videli `./components/SimplePanel`). Vzorčna izvedba SimplePanel je primer enostavnega vtičnika, ki izriše barvni krogec.

Najprej si oglejmo del vrstice, ki izvozi naš vtičnik:

```
new PanelPlugin<SimpleOptions>(SimplePanel)
```

Gre za konstruktor, ki ustvari vtičnik. Podati mu moramo dva parametra, prvi je opis nastavitev, ki jih naš vtičnik podpira, drugi pa gradnik, ki bo izvajal vizualizacijo podatkov.

To vrstico smo spremenili v :

```
new PanelPlugin<WindroseOptions>(WindrosePanel)
```

Zato smo tudi pri uvozu iz `types.ts` dodali `WindroseOptions` ter namesto `SimplePanel` uvozili `WindrosePanel` iz `./components/WindrosePanel`. Najprej si oglejmo, kaj nam pove `WindroseOptions`. Gre za vmesnik (interface), ki opredeli nastavljive lastnosti našega vtičnika. V datoteki `module.ts` vidimo, da se poleg konstruktorja, kliče še ena funkcija:

```
.setPanelOptions((builder) => {  return builder ...
```

V tej funkciji se vrne objekt imenovan `builder`, kateremu lahko dodajamo nastavljive parametre vtičnika s klici funkcij. Med drugimi smo za naš vtičnik uporabili sledeče funkcije:

- addTextInput – za dodajanje tekstovnega polja
- addNumberInput – za dodajanje polja za vnos števila
- addBooleanSwitch – za vklopno stikalo
- addRadio – za radijski gumb (več možnosti med katerimi je lahko izbrana samo ena)
- addSelect – za izbirni seznam
- in še nekatere druge

Vsaka izmed naštetih funkcij prejme JSON objekt kot parameter v katerem so navedene lastnosti kot so opis, ime, vrednosti ter v katero polje (v našem primeru `WindroseOptions`) se naj vnesena vrednost shrani. Te vrednosti so nato dostopne v gradniku.

Lastnosti vtičnika Operato Windrose, ki so nastavljive s pomočjo v uporabniškem vmesniku Grafane, so:

- barvna shema,
- število režnjev,
- število odsekov v posameznem režnju diagrama,
- velikost intervala hitrosti vetra v posameznem odseku režnja,
- število decimalnih mest v pojavnem oknu z opisom stanja (angl. tooltip),
- enota hitrosti vetra (km/h, m/s, mph, ...),
- izbira izpisa oznak (smeri neba ali stopinje),
- prikaz legende.

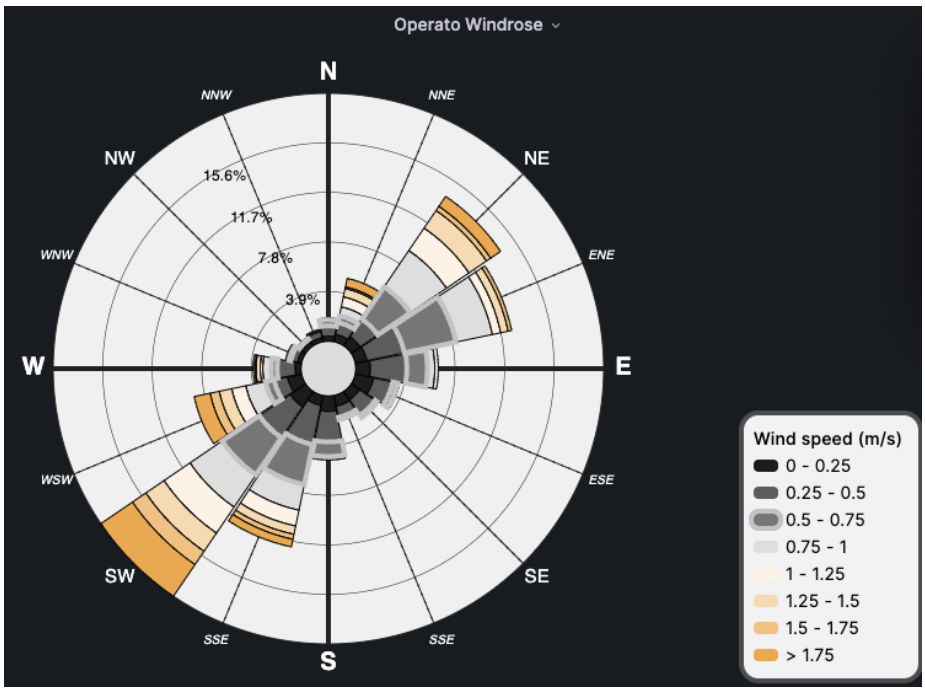
`WindrosePanel` je ReactJS gradnik, ki skrbi za izris našega diagrama roža vetrov. Zamišljen je bil kot vsebnik, ki bo skrbel za preprocesiranje podatkov ter usklajeval pretok podatkov med njegovimi sestavnimi deli. Sam je sestavljen iz dveh delov, ki sta prav tako ReactJS gradnika: Windrose ter WindroseLegend. Kot vtičnik tipa grafikon nima neposrednega dostopa do virov podatkov, zato mu Grafana preko parametrov poda vse potrebne podatke za delovanje. Prejme štiri parametre: nastavitve (options), podatke (data), širino (width) ter višino (height). V nastavitvah so shranjene vrednosti, ki so nastavljene v nastavitvah grafikona s strani končnega uporabnika in se lahko spreminjajo. Podatki do tisti, ki jih je Grafana pridobila iz podatkovnega vira. Širina in višina služita pri vizualizaciji za določanje velikosti komponent znotraj gradnika.

Kot vsi grafikoni v Grafani, je tudi naš Windrose grafikon združljiv z vsemi viri podatkov, ki jih Grafana ponuja. Edini pogoj je, da lahko viru podatkov določimo obliko podatkov, ki jih mora vračati. Grafana namreč ločeno na zalednem delu opravi pridobivanje podatkov, nato pa te podatke zloži v posebno podatkovno strukturo imenovano *Dataframe*. V bistvu gre za dvo-dimenzionalno tabelo podatkov, kjer stolpci predstavljajo vrednost posameznega atributa, vsaka vrstica pa predstavlja en vnos. Ta podatkovna struktura se pošlje k odjemalcu, ki jo posreduje vtičniku za vizualizacijo. Operato Windrose vtičnik predpostavlja, da bo prejeta podatkovna struktura sestavljena

iz dveh stolpcev imenovanih *wind_direction* in *wind_speed*. Vse prejete podatke nato najprej razdeli v režnje glede na vrednost komponente *wind_direction* ter nato še v posamezne odseke znotraj režnjev na podlagi vrednosti komponente *wind_speed*. Zadnji korak predobdelave podatkov je izračun relativnih velikosti režnjev in posameznih odsekov.

WindrosePanel gradnik poskrbi tudi za barvno shemo grafikona. Uporabnik lahko v nastavitvah izbere eno izmed vnaprej pripravljenih barvnih shem, na podlagi katere se gradniki ustrezno obarvajo. Za barvne sheme uporabljamo posebno zmogljivost tehnologije ReactJS imenovano stanja (angl. state) [6]. Omogočajo enostavno proženje ponovnih izrisov gradnikov ob spremembi vrednosti stanja. V našem primeru so stanja barve posameznih odsekov ter barvnih indikatorjev v legendi. Vsem odsekom režnjev ter barvnim indikatorjem v legendi smo dodali poslušalca (angl. listener), ki čaka, da se uporabnik z miško premakne nad ta element uporabniškega vmesnika. Ob tem dogodku se v barvni shemi spremeni barva izbranega odseka, kar se odrazi v spremembi barve vseh odsekov tega vrednostnega razpona (tudi v drugih režnjih) in tudi pripadajočega indikatorju v legendi. Na sliki 8 je prikaz opisanega delovanja stanj, kjer smo šli z miško čez element v legendi, ki zaznamuje interval 0.5 – 0.75.

Izdelava samega gradnika za vizualizacijo je bila izvedena z uporabo HTML značk za vektorsko grafiko. Za krožne črte smo uporabili značko `<circle>` za režnje in odseke smo uporabili značko `<polygon>` ter za ravne črte značko `<line>`. Legenda je izdelana z uporabo HTML `<div>` značk ter z uporabo stilov CSS oblikovana in poravnana v desni spodnji kot starševskega gradnika.



Slika 8: Primer spremembe barve z uporabo stanj.

Uporaba vektorske grafike nam omogoča, da lahko uporabnik grafikon po želji povečuje in zmanjšuje, ob tem pa grafikon ohrani lep izgled brez zamegljenih robov, ki nastanejo pri povečavi rastrskih slik. Sprva smo z značko `<circle>` , ki smo ji nastavili belo barvo polnila, izrisali ozadje grafikona. Nato pa, prav tako z uporabo značke `<circle>`, izrisali črne koncentrične kroge. Število teh krogov je odvisno od absolutne velikosti grafikona znotraj okna brskalnika. Črte, ki kažejo smeri neba, smo izrisali z značko `<line>`, pri čemer smo število črt uskladili z izbrano nastavitvijo števila režnjev grafikona. Vsi tekstovni elementi pa so izrisani z značko `<text>`. Vsem HTML značkam za vektorsko grafiko je potrebno opredeliti, kje in kakšne oblike naj bo izrisan grafični element. Na primer, značka `<line>` zahteva štiri atribute x1, x2, y1 ter y2. Ti atributi definirajo dve točki med katerima naj se izriše črta. Podobno se uporablja značka `<circle>`, kjer moramo definirati točko, ki predstavlja središče kroga in podati polmer kroga. Še bolj enostavna je uporaba značke `<text>`, ki zahteva samo eno točko ter tekst, ki naj ga izpiše. Malo bolj zapletena pa je uporaba značke `<polygon>`. Ta zahteva tekstovni niz v katerem

naštejemo sosledje točk med katerimi se naj izrišejo črte. Primer tašnega niza je: »1,1 2,2 3,2 1,1«. Takšen niz bo iz točke (1,1) narisal črto do točke (2,2), nato od (2,2) do (3,2), ter zaključil črto nazaj do (1,1). Mi smo `<polygon>` značko uporabili za izris odsekov režnjev grafikona. Z uporabo kotnih funkcij ter upoštevanjem pravil za translacijo in skaliranje, smo izračunali položaje točk. Vsak odsek je sprva trapez, ki ga definirajo štiri točke. Nato pa za lepši izgled zgornjo in spodnjo stranico najprej razdelimo na šestnajst delov, novo nastalim točka pa izračunamo položaj tako, da imajo vse enako oddaljenost od centra grafikona. Tako dobimo okrogle odseke režnjev, kot so predstavljeni na sliki **Napaka! Vira sklicevanja ni bilo mogoče najti.**.

5.5 Objava vtičnika

Po končanem razvoju vtičnika si želimo vtičnik prenesti iz razvojne v produkcijsko Grafano, vendar ta zavrne vsak vtičnik, ki ni podpisan. Zato je potrebno najprej zgraditi vtičnik z ukazom:

```
> yarn build
```

Za podpisovanje je potrebno ustvariti račun na www.grafana.com ter slediti navodilom za izdelavo ključa API. Tega moramo pred podpisovanjem shraniti v spremenljivko okolja GRAFANA_API_KEY (to lahko storimo z uporabo ukaza export), nato pa lahko poženemo ukaz:

```
> npx @grafana/sign-plugin@latest --rootUrls http://localhost:3000
```

S tem smo naš vtičnik podpisali z zasebnim varnostnim nivojem. To pomeni, da lahko naš vtičnik namestimo le na Grafane, ki so dostopne na naslovih, ki smo jih navedli z zastavico `--rootUrls` (navedemo jih lahko več, a morajo biti ločene z vejico).

Če pa želimo vtičnik objaviti v uradnem katalogu vtičnikov, pa moramo vtičnik dodatno pripraviti, kot je opisano v navodilih na spletni strani. To vključuje odstranjevanje vseh morebitnih občutljivih ali zaupnih podatkov v izvorni kodi, kot so na primer gesla ali API ključi. Spisati moramo navodila za uporabo vtičnika, ter dodati opis licence, pod katero želimo objaviti vtičnik. Naš vtičnik je objavljen pod licenco Apache License Version 2.0, January 2004. Preveriti moramo tudi samo delovanje vtičnika ter opraviti validacijo vtičnika z orodjem `grafana/plugin-validator-cli`. Grafana ima zelo dobro dokumentacijo, v kateri je podrobno opisan vsak omenjen korak [5].

Ko smo se prepričali, da v vtičniku ni prisotnih podatkov, za katere ne želimo, da do njih dobi dostop širša javnost, lahko na uporabniškem računu na www.grafana.com ustvarimo zahtevo za objavo vtičnika, kjer od nas zahtevajo tri povezave:

- arhiv zip z izdelanim vtičnikom,
- pripadajočo zgoščeno vrednost sha1, ter
- povezavo do izvorne kode (repozitorij git).

Po oddaji zahtevka bo recenzent na Grafani preveril skladnost našega vtičnika in nam dodelil pravico objave vtičnika z javnim varnostnim nivojem. Nato bo od nas zahteval posodobitev zahtevka. Zadnji korak je tako ponovni podpis vtičnika z ukazom:

```
> npx @grafana/sign-plugin@latest
```

Tokrat smo izpustili zastavico `--rootUrls`, kar pove skripti da gre za javni podpis (brez odobrenega varnostnega nivoja podpis ne bi uspel). Recenzent nam nato odobri objavo in v roku 24 ur je vtičnik dostopen na uradnem katalogu vtičnikov Grafane.

Za posodobitev vtičnika se izvede podoben postopek. Izpolni se vlogo za posodobitev različice vtičnika ter odda enake povezave kot pri prvi objavi vtičnika. Le da tokrat vtičnik že prvič podpišemo z javnim varnostnim nivojem.

6 Zaključek

Kompleksne rešitve v IT so toliko dobre, kolikor so dobri njihovi večji in manjši sestavni deli. Velja tudi za varnost, učinkovitost in za vse druge lastnosti. Uporabniki pogosto ocenjujejo celotno rešitev na osnovi nekaj najpogosteje uporabljanih funkcionalnost, ki niso nujno najbolj kompleksni deli rešitve. V primer SUMO je ena od bolj izpostavljenih funkcionalnosti vizualizacija vremenskih podatkov.

Operato nudi svojim strankam pred-pripravljeno vizualizacijo z uporabo Grafane. Grafana je zelo ustrezna komponenta naše rešitve. Je široko uporabljeno orodje v industriji, akademskih krogih in drugih sektorjih, kjer je pomembno vizualizirati in analizirati podatke v realnem času. Zaradi svoje odprtokodne narave je prilagodljiva in omogoča širok nabor razširitev ter integracij z drugimi orodji in platformami. Zaradi vseh naštetih lastnosti je Grafana odlična izbira za vse, ki si želijo prilagodljivo in zmogljivo rešitev za vizualizacijo podatkov.

Prikaz večine podatkov smo enostavno izvedli z privzetimi grafikoni v Grafani, manjkala pa nam je dobra vizualizacije smeri in hitrosti vetra. Ravno veter pa ima največji vpliv na delovanje algoritma in si ga uporabniki zato želijo podrobno analizirati. Ker v analizah podatkov, ki jih za uporabnike izvajamo s programskima jezikoma Python in R, uporabljamo diagram roža vetrov, smo želeli tako prikazati podatke tudi v Grafani. Vendar med obstoječimi vtičniki nismo našli primernega za uporabo v naši rešitvi. Nekateri so bili zastareli, drugi opuščeni sredi razvoja, nekateri pa so kar obstali v zgodnji fazi razvoja. Zato smo se odločili vložiti čas v razvoj lastnega vtičnika za izris diagrama roža vetrov, ki smo ga kasneje objavili tudi javno objavili.

Za izvedbo naloge je bilo potrebno osnovno poznavanje jezika TypeScript, tehnologije ReactJS in formata SVG za opis vektorske grafike. Vsa uporabljena orodja so prosto dostopna.

Podjetje Operato svoje rešitve gradi na odprtokodnih in prosto dostopnih tehnologijah in orodjih. Z izvedbo in javno objavo vtičnika za napredno vizualizacijo vremenskih podatkov v Grafani smo se sedaj skupnosti oddolžili s pripravo zanimivega izdelka, ki ga lahko kdorkoli vključi v svoje rešitve. Ob času pisanja tega članka, nekaj manj kot mesec dni po objavi prve različice vtičnika, je na uradnem katalogu vtičnikov Grafane zbral že več kot 500 prenosov.

Literatura

[1] KOSEC Gregor, MAKSIĆ Miloš, DJURICA Vladimir. »Dynamic Thermal Rating of Power Lines – Model and Measurements in Rainy Conditions«. International Journal of Electrical Power & Energy Systems, 2017.

[2] MEOLIC Robert, LENKO Miha, SOUVENT Andrej. »Napredne IT rešitve za pospeševanje zelenega prehoda«. In Proc. OTS 2022, pp. 206-217, 2022.

[3] Roža vetrov. Wikipedija. https://sl.wikipedia.org/wiki/Roža_vetrov

[4] Wind rose. Wikipedia. https://en.wikipedia.org/wiki/Wind_rose

[5] Grafana Labs. Grafana plugin developer's guide. https://grafana.com/docs/grafana/latest/developers/plugins/

[6] Meta Open Source. Built-in React Hooks. https://react.dev/reference/react