Power BI liidesele loodud ReactJS teegi täiustamine

Sisukord

[Sissejuhatus 3](#_Toc6218148)

[Probleem 3](#_Toc6218149)

[Eesmärk 3](#_Toc6218150)

[Metoodika 3](#_Toc6218151)

[Allikad 3](#_Toc6218152)

[Sõnastik 4](#_Toc6218153)

[Metoodika 5](#_Toc6218154)

[Ülevaade objektist 5](#_Toc6218155)

[Ülevaade tööriistadest 5](#_Toc6218156)

[Ülevaade protsessist 5](#_Toc6218157)

[Validatsioon 5](#_Toc6218158)

[Verifikatsioon 5](#_Toc6218159)

[Tulemused 6](#_Toc6218160)

[Tüüpimine 6](#_Toc6218161)

[Lisatud funktsionaalsus 6](#_Toc6218162)

[Eritüüpi infograafid 6](#_Toc6218163)

[Täiendatud sätted 6](#_Toc6218164)

[Kohandatavad filtrid 6](#_Toc6218165)

[*PageName* 6](#_Toc6218166)

[Järjehoidjad 6](#_Toc6218167)

[Overriding Error Experience 7](#_Toc6218168)

[Testimine 7](#_Toc6218169)

[Disainimuutused 7](#_Toc6218170)

[Analüüs 9](#_Toc6218171)

[Mida teeksin paremini? 9](#_Toc6218172)

[Mida teha tulevikus? 9](#_Toc6218173)

[Kokkuvõte 10](#_Toc6218174)

# Sissejuhatus

## Probleem

Motivatsioon bakalaureusetööks tuleneb puudulikust integratsioonist graafilise liidese arendusvahendi *ReactJS*-i ning ärianalüüsi ja visualiseerimistarkvara *Power BI* vahel. Arendades veebiaplikatsiooni, mis lülitab endasse *Power BI* funktsionaalsust, on vaja kasutada *Microsofti* pakutud *JavaScripti* teeki *powerbi-client*. Tuginedes vaid mainitud moodulile, on *Reacti* arendaja aga sunnitud lahti ütlema komponent-disainist, mis *React*-iga käsikäes käib. *React­*-ile on arendatud Power BI ühilduseks mitmeid teeke **(viide)**, ent töö kirjutamise hetkel jätavad nad palju soovida: vajalikku funktsionaalsust on jäetud implementeerimata, puuduvad testid, puuduvad tüüpimisfailid arendajatele, kes kirjutavad koodi *TypeScript*-iga.

## Eesmärk

Töö eesmärgiks on pakkuda Reacti arendajale suuremaid võimalusi nign mugavust Power BI kasutamisel üheaegselt *powerbi-client-*i ennast otseselt rakendamata. Implementeeritakse järgnevad funktsionaalsused:

* Filtrite seadmine ja rakendamine
* Järjehoidjate seadmine ja rakendamine
* Avavaate seadmine
* Liigse info varjamine

## Metoodika

Arendustöö raamistikuks on *Facebook*-i loodud *ReactJS*. Aluseks võetakse *react-powerb*-i teek, millele ehitatakse edasist funktsionaalsust. Koodikirjutamisel lähtutakse *Behaviour Driven Development* põhimõtetest. Enne funktsionaalsuse implementeerimist luuakse vastavad testid, mis kirjeldavad endas nõudeid lisatavale koodile. Teste luuakse *AirBNB Enzyme* teegi abil.

## Allikad

 Ferrari, Alberto and Marco Russo. *Introducing Microsoft Power BI.* Redmond: Microsoft Press, 2016.

 Beck, Kent. *Test-Driven Development : by example.* Boston: Addison-Wesley, 2003.

 Sherman, Rick. *Business Intelligence Guidebook :From Data Integration to Analytics*. Amsterdam: Morgan Kaufmann, 2015.

 Facebook Inc. 2018. *React a JavaScript Library for Building User Interfaces.* [ONLINE] Saadaval: https://reactjs.org/.

# Sõnastik

Järgnev sõnastik on loodud töö hõlpsamaks mõistmiseks ning loetavuse parandamiseks.

* *rendering –* definitsioon. Edaspidi “visualiseerimine”
* *npm* – definitsioon.
* *github –* definitsioon.
* *JavaScript –* defintisioon. Edaspidi “*JS”*
* *TypeScript –* defintisioon. Edaspidi “*TS”*
* *DOM –* definitsioon.
* *React lifecycle method* – definitsioon. Edaspidi “elutsüklimeetod”
* *BDD* – definitsioon.

# Metoodika

Järgnevalt antakse ülevaade uuritavast objektist ning tööriistadest ja protsessist, millega töö lõpptulemus saavutati.

## Ülevaade objektist

*React-powerbi* on *npm-*ist päritav teek, mille avatud lähtekoodiga projekti leiab *github­*-ist. (AADRESS) Tegemist on *ReactJS* teegiga, mille funktsioon on veebiteenustesse integreerida *PowerBI*-s loodud andmevisualiseerimisobjekte (edaspidi visuaale). Oma praeguses vormis on *react-powerbi* disainitud kasutajale, kes ei soovi visuaalidega suhelda ega neid manipuleerida, vaid neid ainult puhtal kujul lõpptarbijale edastada. Soovides teha muudatusi juba integreeritud visuaalile, tuleb integreerivale komponendile edastada uuendatud parameetrid, mis käivitavad aga komponendis uuesti ajakuluka visualiseerimisprotsessi.

Teek toetub oma implementatsioonis täielikult *Microsoft*-i loodud *PowerBI-JavaScript* teegile, mis pakub meetmeid visuaalide ja nendega seotud teabe pärimiseks, integreerimiseks ning manipuleerimiseks. Kasutades *PowerBI- JavaScript* pakutud meetode on võimalik suhelda integreeritud visuaaliga ilma, et komponendile peaks uusi parameetreid pakkuma. Seeläbi saab ära hoida ebavajalikku komponendi visualiseerimist, mis on aeganõudev ning takistab lõpptarbija tegevust.

Käesoleva töö kaudu uuendatud teek on esmajoones loodud et

Too sisse *Client owns data vs App owns data* põhimõtted et näidata, et loodud töö on mõeldud *App owns data* kasutajatele.

## Ülevaade tööriistadest

Arendustööks kasutatakse *IntelliJ IDEA Ultimate Edition*-it.

Teste kirjutatakse *Mocha* raamistiku ning *chai, Enyzme-BNB* ja *sinon* teekide abil.

Töö on koostatud tekstiredigeerimise aplikatsiooniga *Microsoft Word*.

Miks neid kasutasid? Mis olid alernatiivid?

## Ülevaade protsessist

Arendustöö jaotub neljaks osaks: 1) Ümberdisainimine 2) Testimine 3) Funktsionaalsuse lisamine 4) Tüüpimine. Algne *react-powerbi* teek kasutab tulemuse saavutamiseks mitmeid kaheldava funktsionaalse väärtusega võtteid. Ümberdisainimise eesmärgiks on baasteeki refaktoreerida muutes seekaudu koodi loetavamaks ja asjalikumaks. Testimine ja uue funktsionaalsuse lisamine käivad käsikäes kasutades *BDD*-d, et luua koodibaas, mis on tulevaste muutuste puhul kontrollitav ning kergelt adapteeruv. Valmis arendusele saab lõpetuseks kirjutada tüüpimisfaili, mis pakub suuremat kasutuslihtsust *TS*-i arendajatele.

### Validatsioon

Töö baseerub kahte tüüpi validatsioonile. Projekti tüüpimist ning testidega varustamist on palunud oma *TODO* nimistus *react-powerbi* looja. (VIIDE) Uue funktsionaalsuse lisamise õigustamine toetub *PowerBI-Embedded* kasutusjuhtudele ja implementatsiooni kirjeldustele, mis *Microsoft* on oma veebikeskkondades välja andnud. Seega on eeldatud, et tegemist on vajaliku funktsionaalsusega, mille validatsioon on juba tehtud.

The definition of Validation according to IEEE-STD-610 is:

“An activity that ensures that an end product stakeholder’s true needs and expectations are met.”

### Verifikatsioon

Töö verifitseerimiseks on arenduse käigus kasutatud *BDD* põhimõtteid. Igat lisatud funktsionaalsust saadab vähemalt kaks testi, mis kirjeldavad, kuidas süsteem peaks toimima ning kuidas mitte.

Software Engineering standards known as IEEE-STD-610 defines “Verification” as:

“A test of a system to prove that it meets all its specified requirements at a particular stage of its development.”

# Tulemused

## Tüüpimine

Teegi tüüpimine oli lisaks testidele tarkvara looja põhilisi soove, mille lisamisel arvab ta heaks avaldada teegi esimse ametliku versiooni. (VIIDE) Nimelt lisab tüüpimine JS koodile loetavust ja arendusmugavust, sest arenduskeskkond oskab juba enne kompileerimist välja näidata vääratest tüüpidest tulenevaid vigu ning juhendada arendajat tüübitud funktsionaalsuse kasutamisel. Samal ajal kompileerub tüübitud kood ikkagi puhtaks JS-iks, nii et programmi jooksukiirus ei muutu. (VIIDE) Arvestades, et [protsent] arendajaid kasutab TS-i, on tüüpimine põhjendatud.

Kuigi lähtekood pole kirjutatud TS-is, on võimalik ikkagi tarbijatele tüüpimist pakkuda. Deklareeritakse ning eksporditakse liides, mille kasutajatele jääb mulje, et suheldakse TS, mitte JS koodiga.

Tüüpimist hõlbustab *npm*-is saadaval projekt *DefinitelyTyped*, mis lubab lisada teiste inimeste JS teekidele tüübitud liideseid ilma, et neil peaks olema õigus mainitud teekide lähtekoodile. Kui liidesele teha pull request ning see vormi poolest õigeks tunnistatakse, luuakse tüübile automaatselt @types-algusega npm-projekt, mida kasutajad koos defineeritud mooduliga kasutada saavad. Uurimustöö kirjutaja otsustas aga kirjutada tüüpfailid otse teegi projekti, kuna originaalne autor oli avaldanud soovi kaasata .d.ts fail oma moodulisse.

## Lisatud funktsionaalsus

Enne uue funktsionaalsuse lisamist tuleb otsustada, kuidas ning mis tingimustel toimub *PowerBI-Embedded* ja *react-powerbi* vaheline suhtlus. Integreeritud objekte saab mõjutada kolmel moel: 1) konfiguratsiooni objektiga 2) visuaaliobjekti meetoditega enne visualiseerimist 3) visuaaliobjekti meetoditega pärast visualiseerimist.

Konfiguratsiooniobjekti modifitseerimine on kõige lihtsam ja ainus variant, mida algne *react-PowerBI* pakub: luuakse konfiguratsioon, mis edastatakse koos integreeritava *DOM-*i elemendiga integreerivale meetodile ning vastuseks luuakse ning seotakse pakutud elemendiga *PowerBI* visuaal. See lähenemine sobib visuaali algseks loomiseks, ent kui konfiguratsiooni muuta, visualiseeritakse visuaal uuesti nullist.

Visuaalobjekti visualiseerimist on võimalik jaotada mitmeks osaks, kasutades *PowerBI-JavaScript-*I *Phased Embedding API-*t. See võimaldab objekti alguses alla laadida, pärida sellelt andmeid (näiteks filtreid ja järjehoidjaid), muuta objekti olekut saadud andmete põhjal ning alles seejärel objekti visualiseerida. Seega on antud võte tarvilik, kui vajalik info tuleb teenuselt ning visualiseerimine on vaja teha kasutajale märkamatult.

Kolmas variant kätkeb endas visuaalobjekti meetodite kasutamist pärast visualiseerimist. Vajalik siis, kui arendaja tahab tingimis muuta visuaalobjekti olekut, samas varjates kasutaja eest *PowerBI-Embedded* pakutud *tab*-e.

Kuidas otsustada, milliseid variante kasutada? Silmas pidades, et ei tekiks funktsionaalsuse ülesatureerimist.

Konfiguratsiooniobjekti variandi saavad kõik funktsionaalsused, et oleks võimalik sisuliselt kõike kasutada puhtalt komponendi parameetried kasutades.

Teise ja kolmanda variandi saavad vaid need, kellel pole lihtsalt true/false väärtused ja keda võib tahta manipuleerida töö käigus. Kui võimalik, siis varianti nr 2 ei kasuta, et hoida kood lühike.

Setterid paljastatakse sellisele funktsionaalsusele nagu *Bookmarks, Filters, PageNames*.

### Eri tüüpi visuaalid

Lisaks raportitele on teegiga võimalik kinnitada saidile ka *Dashboard*-e ning *Tile*-e.

Millised on parameetrilised erinevused?

### Täiendatud sätted

* Lokaal
* Nähtavus
* Järjehoidjate paneel

### Kohandatavad filtrid

### *PageName*

### Järjehoidjad

Praegusel hetkel *getBookmarks()* ei tööta embeddimata objektiga.

Miks on see minu jaoks tähtis? (sest me ei taha 1) kliendile näidata üleliigset infot 2) madaldada kliendi kindlustunnet sellega, et ekraanil info liigub ilma nende sisendita)

Järjehoidjad lubavad arendajal kuvada kasutajale täpselt selliseid andmeid nagu vaja, samal ajal jubaloodud raporte taaskasutades, meenutades oma funktsioonilt seega filtreid. Erinevalt aga filtritest, mida saab teades käsitletavaid andmevälju koostada ka kliendipoolt, on järjehoidjaid vaja enne kasutamist ettenähtud keskkonnas raportile lisada.

Märkamatuks muutmiseks kasutati *powerBI-JS* pakutud kaheosalist *embed*-imist. Algul küsitakse raporti objekt, sellelt saab seejärel pärida või sellele rakendada väärtuseid ning objekti *embed*-imine toimub viimasena.

Sõltuvalt rakendatud propsist saab kas 1) pärida nime järgi (otse) 2) pärida nime järgi (load) 3) pärida järjekorranumbriga (load)

Milleks variant 1? Tundub pointless arvestades mu kasutusjuhte.

Kas bookmarki funktsionaalsus liigutada Raportisse? Arvestades, et (vist) ainult raportid saavad kasutada bookmarke.

### Overriding Error Experience

## Testimine

Testid jagunevad kaheks. Ühed kasutavad *PowerBI-Javascript-*i valideerimismehhanisme ja kontrollivad, kas väärate või puudulike konfiguratsiooniobjekti parameetrite puhul heidetakse korrektne veateade. Teised hindavad olekumuutuseid visuaalobjekti pakutud funktsioonide kutsumisel.

## Disainimuutused

Miks tegin klassist funktsiooni?

*PowerBI-react* on oma sisulisele lihtsusele vaatamata liiga keeruliselt üles ehitatud. Seda ümber tehes lähtuti antud põhimõttetest: a) Liigse eemaldamine b) Loetavuse parandamine

Liigset vältides eemaldati järgnevad *React*-i elutsüklimeetodid: *componentWillReceiveProps, componentDidUpdate, componentWillUnmount*.

***componentDidMount***

***PS! Seda ei eemaldatud, vaid see pandi useEffects hooki abil koos componentDidUpdateiga kokku.***

Funktsiooni kutsutakse komponendi elutsükli jooksul vaid korra – pärast esimest renderdust.

***componentWillReceiveProps***

Funktsiooni kutsutakse, kui *mount*-itud komponent saab uued *props*-id, aga enne, kui nende põhjal komponendi kuva uuendatakse. Selle kasutamine on põhjendatav, kui on vaja sissetulevaid parameetreid kontrollida ja ümber kirjutada enne, kui nad komponendis muutusi esile toovad. Integreeritud PowerBI komponent ei renderda ennast aga otseselt olekusmuutusest ega uutest parameetritest tingituna, vaid siis, kui kutsutakse otseselt *embed()* meetodit. **Testin, kuidas süsteem praegu töötab, kui uuendan configi.** Vastus: ei tööta paremini, iframe ikkagi renderdab ennast uuesti. (Pane koodinäide.)

***componentWillUnmount***

Funktsiooni kutsutakse, kui komponent eemaldatakse DOM-ist ning on mõeldud selleks, et lõpetada *subscriber* tüüpi meetodeid, et need asjata taustal edasi ei käivituks. Varasemas järgus kutsuti selles funktsioonis *powerbi-js Service* objektilt *reset()* meetodit, mis haagib genereeritud *iframei* lahti vastavast DOM elemendist. See on aga üleliigne, sest vastav element on genereeritud samas komponendis mis iframe ning komponendi *unmount*-imisel hävivad kõik tema lapsed. (VIIDE)

Välja jäeti parameetrid *height* ja *width*, sest elementi suurust tingivad kasutajad ise *CSS*-i kaudu (VIIDE). Et garanteerida sujuv integratsioon potentsiaalse *CSS*-iga, anti *Embedder* komponentile püsistiil, mis sunnib teda alati olema sama suur vanemkomponendiga. See aitab eraldada üksteisest funktsionaalsuse ning stiliseerimise, luues koodikirjutajale selgemad arendustingimused.

Komponendist eemaldati osa koodist, mis uute parameetrite ilmnemisel kopeerib ümber vaid muutunud väärtused. Võib oleteaga, et sellist lähenemist kasutatakse koormuse vähendamiseks komponendi taasvisualiseerimisel. Kuna kõik parameetrid lähevad aga *PowerBI* konfiguratsioonfaili, ei oma selline disain siinkohal tähtsust. (VIIDE ja põhjendus toetudes Reacti renderdus põhimõttetele)  
  
Defaultprops: kas jätta undefined või teha ternary loogika? Ternary loogika, kuigi tõenäoliselt ebavajalik, muudab koodi tulevikus loetavamaks. Teisalt kui undefined loogika peaks kunagi muutuma?

# Analüüs

## Mida teha paremini?

Peale etteantu (tüübid ja testid) on nõudlus pärit Microsofti kasutusjuhtudest. Tulevikus tuleks teegi pakutavad teenused kindlasti otsese nõudlusega vastavusse panna.

Kui ma poleks teinud vahele extra komponente (Report, Dashboard, Tile), peab uute parameetrite kasutamiseks teeki muutma.

## Mida teha tulevikus?

Kahepoolne suhtlus. Et kui on autoriseeritud kasutaja saab too teha ka teatud jäävaid muutuseid (nt isiklikud järjehoidjad)

## Kas midagi sarnast on tehtud?

Microsoftil oli endal varem reacti ja powerbi teek. Enam mitte. Uuri miks? Mida see täpsemalt pakkus?

Kuidas näevad välja teiste *frontend* raamistikke (Angular, Vue) powerbi integratsioonid? On need olemas?

# Kokkuvõte