Power BI liidesele loodud ReactJS teegi täiustamine

Sisukord

[Sissejuhatus 3](#_Toc3758357)

[Probleem 3](#_Toc3758358)

[Eesmärk 3](#_Toc3758359)

[Metoodika 3](#_Toc3758360)

[Allikad 3](#_Toc3758361)

[Metoodika 4](#_Toc3758362)

[Ülevaade objektist 4](#_Toc3758363)

[Ülevaade tööriistadest 4](#_Toc3758364)

[Ülevaade protsessist 4](#_Toc3758365)

[Validatsioon 4](#_Toc3758366)

[Verifikatsioon 4](#_Toc3758367)

[Tulemused 5](#_Toc3758368)

[Analüüs 6](#_Toc3758369)

[Kokkuvõte 7](#_Toc3758370)

# 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 *Test 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/.

# Metoodika

## Ülevaade objektist

Kelle jaoks teek on ja kelle jaoks mitte? (On praegu arendatud ühepoolsena, st kasutaja ei saa teha jäävaid muudatusi)

## Ülevaade tööriistadest

## Ülevaade protsessist

### Validatsioon

### Verifikatsioon

# Tulemused

## Tüüpimine

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

Tüüpimist ei morjenda asjaolu, et aluskood pole kirjutatud TS-is. Deklareeritakse ning eksporditakse liides, mille kasutajatele jääb mulje, et suheldakse TS, mitte JS koodiga.

Tüüpimist hõlpustab 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

Infograafide üle antakse rohkem võimu. Embed objekte saab mõjutada 1) konfiguratsiooni objektiga 2) infograafiobjekti meetoditega enne renderdamist 3) -//- pärast renderdamist.

Argumenteeri nende kolme eelised. Kas ma kasutan kõiki?

Kofiguratsiooniobjekti sättimine on kõige lihtsam ning ainus variant, mida algne *react-PowerBI* pakub. Arvestades selle teegi populaarsust võib öelda, et selline lähenemine katab enamus kasutajate vajadused.

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

### Eritüüpi infograafid

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

### Täiendatud sätted

* Lokaal
* Nähtavus

### 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 taaskasutades infograafikuid, meenutades selle poolt filtrite funktsiooni. Erinevalt filtritest aga on vaja infograafi enne ettenähtud keskkonna kaudu redigeerida, et vajalikud järjehoidjad oleksid saadaval.

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.

Ei saa küsida bookmarke enne renderdamist??

Kuhu bookmarki funktsionaalsus kirjutada? Arvestades, et (vist) ainult raportid saavad kasutada bookmarke.

### Overriding Error Experience

## Testimine

Teegi testimist soovis react-powerbi arendaja. Testide üldine iseloom on kontrollida kasutataud valideerimismehhanisme ja jälgida, kas väärate parameetrite puhul heidetakse korrektne veateade.

## Disainimuutused

Miks tegin klassist funktsiooni?

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

Liigset vältides eemaldati järgnevad React elutsükli meetodid: *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)

Mida .reset() teeb? “*Given an HTML element that has a component embedded within it, removes the component from the list of embedded components, removes the association between the element and the component, and removes the iframe.”*

Arvestades, et Embedder genereerib ise divi, mille külge iframe läheb, siis kas reset pole mitte üleliigne?

Uuri, mis täpselt Reacti komponendiga juhtub, kui ta elu lõpeb.

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.

Komponendist eemaldati osa koodist, mis asendab parameetritest vaid muutunud väärtused. Sellist lähenemist kasutatakse, et vähendada koormust komponentide taasrenderdamisel. Kuna kõik parameetrid lähevad aga *PowerBI* konfiguratsioonfaili, ei oma selline disain siinkohal tähtsust. (VIIDE)

# Analüüs

## Mida teeksin paremini?

Peale etteantu (tüübid ja testid) on nõudlus pärit MicroSofti kasutusjuhtudest. Kuna need on esile toodud, olen eeldanud, et tegemist on vajaliku funktsionaalsusea ning et validatsioon(?) on nende institutsiooni poolt juba tehtud. Tulevikus tuleks teegi pakutavad teenused kindlasti otsese nõudlusega vastavusse panna.

## Mida teha tulevikus?

Kahepoolne suhtlus. Et kui on autoriseeritud kasutaja saab too teha ka teatud muutuseid (nt. luua oma bookmarke kasutades *Base64* state-i).

# Kokkuvõte