SAVONIA

OPINNÄYTETYÖ - **AMMATTIKORKEAKOULUTUTKINTO**
TEKNIIKAN JA LIIKENTEEN ALA

LIGHTNINGCHART JS PALVELINPÄÄSSÄ RENDERÖINTI

Opinnäytetyö

TEKIJÄ: Ilkka Kuosmanen

SAVONIA-AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Tiivistelmä

Koulutusala Tekniikan ja liikenteen ala	
Koulutusohjelma/Tutkinto-ohjelma Sähkötekniikan tutkinto-ohjelma	
Työn tekijä(t) Ilkka Kuosmanen	
Työn nimi LightningChart JS Palvelinpäässä Renderöinti	
Päiväys 6.6.2020	Sivumäärä/Liitteet 20
Ohjaaja(t) Yliopettaja Arto Toppinen, Lehtori Pasi Lepistö	
Toimeksiantaja/Yhteistyökumppani(t) Arction Oy, Pasi Tuomainen, Toimitusjohtaja	

Tiivistelmä

LightningChart JS on selaimille kehitetty datavisualisointikirjasto. Opinnäytetyön tavoitteena oli toteuttaa Node JS yhteensopivuus, jotta LightningChart JS -kirjastolla voisi renderöidä palvelimella. Tarkoituksena on laajentaa tuotteen mahdollisia käyttökohteita.

Työssä käydään läpi eri tekniikkavaihtoehtoja yhteensopivuuden toteuttamiseen sekä mitä ongelmia selainpohjaisen kirjaston siirtäminen Node JS -alustalle aiheuttaa. Työssä toteutetaan Node JS -tuki LightningChart JS -kirjastolle sekä esimerkkipalvelin, joka toimii yhtenä sovellus esimerkkinä LightningChart JS:n käytöstä Node JS -alustalla.

Opinnäytetyön lopputuloksena saatiin yhteensopivuus Node JS -alustalle tekemällä Node JS:ää varten yhteensopivuus kirjasto, @arction/lcjs-headless. Työn tulos julkaistiin npm -rekisteriin sekä avoimena lähdekoodina GitHubiin.

Avainsanat
JavaScript, WebGL, TypeScript, Node JS

SAVONIA UNIVERSITY OF APPLIED SCIENCES

THESIS
Abstract

Field of Study Technology, Communication and Transport			
Degree Programme Degree Programme in Electrical Engineering			
Author(s) Ilkka Kuosmanen			
Title of Thesis LightningChart JS server-side rendering			
Date	June 6, 2020	Pages/Appendices	20
Supervisor(s) Mr. Arto Toppinen, Principal Lecturer Mr. Pasi Lepistö, Senior Lecturer			
Client Organisation /Partners Arction Ltd. Mr. Pasi Tuomainen, CEO			

Abstract

LightningChart JS is a browser-based data visualization library. The goal for this thesis was to implement Node JS support for LightningChart JS so that it can be used to render on server-side. The purpose is to add more use cases where the product can be used.

Different methods for implementation and different problems for migrating a browser-based library to Node JS are considered. Node JS support for LightningChart JS library is implemented in this thesis. Also, an example server is created to act as one usage example for LightningChart JS use with Node JS.

As a result of this thesis, compatibility with Node JS for LightningChart JS is achieved by making a separate compatibility library, @arction/lcjs-headlss. The results were published to npm registry and as an open source code to GitHub.

Keywords
JavaScript, WebGL, TypeScript, Node JS

SISÄLTÖ

1 JOHDANTO

Opinnäytetyön tilaajana toimii Arction Oy. Arction on datavisualisointiin erikoistunut ohjelmistoalan yritys. Päätuotteita ovat LightningChart .NET ja LightningChart JS. Molemmat tuotteet ovat datavisualisointikirjastoja, LightningChart .NET on C# pohjainen .NET alustalle kehitetty kirjasto ja LightningChart JS on monialustainen JavaScript pohjainen kirjasto. LightningChart JS tukee monia eri alustoja moderneista selaimista aina älypuhelinsovelluksiin.

Opinnäytetyön tavoitteena on lisätä yksi uusi käyttöalusta LightningChart JS tuotteeseen. Tavoitteena on pystyä käyttämään olevassa olevaa tuotetta Node JS -alustalla, joka mahdollistaa kirjaston käytön datavisualisointiin palvelimilla ja erilaisissa Node JS -ympäristöissä toimivissa sovelluksissa. LightningChart JS käyttää WebGL teknologiaa, jolle ei ole suoraan olemassa tukea Node JS alustalla. Tämän takia työ aloitetaan tutkimalla eri vaihtoehtoja, miten Node JS -tuki on mahdollista saavuttaa. Löydetyistä vaihtoehdoista valitaan yksi, jota käyttämällä Node JS -tuki luodaan.

2 LIGHTNINGCHART JS

LightningChart JS on Arction Oy:n kehittämä WebGL -pohjainen kaaviokirjasto. (Arction Oy, 2020) LightningChart JS on kehitetty selainpohjaisia ympäristöjä varten ja sitä pystyykin tällä hetkellä käyttämään Electron-, Android-, ja IOS-sovelluksissa sekä selaimissa.

Kuva 1. Osakearvojen muutosten visualisointi LightningChart JS -kirjaston avulla. (Arction Oy, 2020)

LightningChart JS -kirjasto mahdollistaa interaktiivisten ja reaaliaikaisten kaavioiden luomisen. Sillä pystyy visualisoimaan kymmeniä miljoonia datapisteitä reaaliaikaisesti. (Arction Oy, 2020)

3 TEORIA

3.1 Lyhenteet ja määritelmät

WebGL	Verkkografiikka kirjasto korkean suorituskyvyn 3D ja 2D grafiikan käyttöön verkkoselaimissa. (MDN Web Docs, 2020)
Node JS	JavaScript moottori, joka on suunniteltu skaalattavissa olevien verkkosovellusten rakentamiseen. (OpenJS Foundation, 2020)
Renderöinti	Kuvien piirtäminen puskuriin.
JavaScript	Ohjelmointikieli, joka on eniten tunnettu verkkosivujen ohjelmointi kielenä. (MDN Web Docs, 2020)
TypeScript	JavaScriptiin päälle kehitetty ohjelmointikieli, joka käännetään JavaScriptiin. (Microsoft, 2020)
ANGLE	Almost Native Graphics Layer Engine, Grafiikka -rajapinta, joka kääntää OpenGL ES rajapintakutsut laitteistokiihdytettyyn rajapintaan. (Google, 2020)
DOM	Document Object Model, verkkosivua kuvaava malli. (MDN Web Docs, 2020)

3.2 JavaScript

JavaScript on alun perin luotu verkkosivujen interaktiivisuuden toteuttamiseen. Nykyään JavaScriptiä on mahdollista käyttää selaimen ulkopuolella Node JS ja Electron alustojen kautta.

3.2.1 Node JS

Node JS:n avulla on mahdollista kirjoittaa palvelinsovelluksia JavaScriptillä. Tämä mahdollistaa saman ohjelmointikielen käytön selain- ja palvelinympäristöissä. Node JS -sallii myös pääsyn järjestelmän ominaisuuksiin, mitkä eivät ole saatavilla selainympäristössä. Tällaisia ominaisuuksia ovat esimerkiksi tiedostojärjestelmä, prosessien hallinta ja käyttöjärjestelmän tiedot. Node JS -käyttää Chromen V8 JavaScript moottoria JavaScriptin suorittamiseen. (OpenJS Foundation, 2020)

3.2.2 JavaScript selaimessa

JavaScript on yleisin tapa lisätä toiminnallisuutta verkkosivuille. (MDN Web Docs, 2020) Viime vuosina JavaScriptin kehitys on mahdollistanut monipuolisempien ja vaativampien sovelluksien kehittämisen selainympäristöön. Selainympäristö on tosin monin puolin rajoitetumpi ympäristö kuin Node JS -ympäristö.

3.2.3 WebGL

WebGL on selaimia varten kehitetty grafiikkakirjasto. Se on kehitetty seuraten OpenGL ES 2.0 -ohjelmointirajapintaa, joka on grafiikkastandardi sulautetuille laitteille. (MDN Web Docs, 2020) WebGL:stä on olemassa myös WebGL 2 -versio, joka toteuttaa OpenGL ES 3.0 ohjelmointirajapinnan. (MDN Web Docs, 2020) LightningChart JS käyttää WebGL 1 -versiota.

4 PALVELIMELLA RENDERÖINTI

Palvelimella renderöimällä pyritään mahdollistamaan enemmän mahdollisia käyttökohteita LightningChart JS -kirjastolle. Se mahdollistaa esimerkiksi PDF-muotoiseen automaattisesti luotuun raporttiin kuvaajan luomisen automaattisesti. Vaihtoehtoisesti se mahdollistaa staattisten kuvaajien luomisen siten, että käyttäjälle näytetään pelkkä kuva kuvaajasta eikä käyttäjän selaimen tarvitse pystyä käsittelemään kaikkea dataa, joka kuvaajan piirtämiseen vaadittaisiin.

4.1 Haasteet

LightningChart JS -on luotu toimimaan selainympäristössä, joten se hyödyntää ominaisuuksia, jotka ovat olemassa vain selaimissa eikä Node JS -ympäristössä. Yksi tällainen ominaisuus on teknologia, jolla kuvaajat piirretään selaimessa, WebGL.

Tämän lisäksi LightningChart JS -käyttää vain selaimissa olevaa Window -ohjelmointirajapintaa. Window -rajapinta on selaimissa käytettävissä globaalin muuttujan ”window” kautta. Tämä rajapinta tarjoaa pääsyn selaimen ohjelmointirajapintoihin ja selaimeen ladattuun DOM-dokumenttiin. (MDN Web Docs, 2020) Palvelimella renderöinnin tukemiseksi molemmat WebGL ja Window -rajapinnat tulee korvata Node JS yhteen sopivilla rajapinnoilla.

Puuttuvien rajapintojen lisäksi, LightningChart JS -pyrkii animoimaan datan lisäämisen ja kuvaajien käsittelyn. Node JS -ympäristössä, missä tavoitteena on saada yksi kuva jossa, on kaikki kuvaajalta haluttu informaatio, mahdollisimman nopeasti, tämä aiheuttaa ongelman. Jos kuva otetaan heti kun data on lisätty, saadaan ulos kuva, jossa datasta ei näy kuin pieni osa. Tämän ongelman ratkaisemiseksi lisätään LightningChart JS -kirjastoon ominaisuus, joka mahdollistaa kaikkien animaatioiden käytöstä poistamisen, kun kuvaaja luodaan.

4.2 Tavoitteet

Tavoitteena on kehittää LightningChart JS -kirjastoon tuki Node JS -alustalle. Tämä mahdollistaa kaavioiden luomisen palvelimen päässä osana Node JS -pohjaista palvelinta. Lisäksi on mahdollista kehittää muita sovelluksia, jotka toimivat Node JS -alustalla, mutta eivät ole palvelimia.

Tärkeimmät huomioalueet:

- Luotettavuus
- Visuaalinen laatu
- Suorituskyky

5 TEKNIIKKA

5.1 Tekniikkavaihtoehdot

Ennen kuin Node JS -renderöinti tukea pystyi kehittämään, piti selvittää mitä vaihtoehtoja puuttuville ohjelmointirajapinnoille on olemassa.

Tärkeimpinä valinta kriteereinä voidaan pitää seuraavia asioita:

- Ylläpidettävyys
- Suorituskyky
- Toteutuksen laajuus

5.1.1 Canvas -ohjelmointirajapinta

Selaimet tarjoavat Canvas -ohjelmointirajapinnan. Tämä rajapinta on tarkoitettu lähinnä 2D-grafiikan renderöintiin. (MDN Web Docs, 2020) Canvas -rajapinta on huomattavasti yksinkertaisempi käyttää kuin WebGL rajapinta, mutta myös hitaampi monissa operaatioissa. Node JS ympäristössä Canvas -rajapinta on mahdollista tuoda node-canvas nimisestä paketista. Tämä paketti toteuttaa Canvas -rajapinnan hyödyntämällä Cairo -grafiikkarajapintaa. (Automattic, 2020)

Jos LightningChart JS:n renderöinti olisi tehty Canvas -rajapintaa vasten niin silloin tällä paketilla olisi pystynyt korvaamaan Node JS -ympäristöstä puuttuvan tuen. Koska näin ei kuitenkaan ole, niin Canvas -rajapinnan käyttö vaatisi uuden renderöintimoottorin kehittämisen. Canvas -rajapintaa kuitenkin käytetään LightningChart JS:ssä joihinkin piirtoihin, joten Canvas -rajapinta pitää toteuttaa myös Node JS -alustalla, joten tämä paketti otetaan käyttöön näiden piirto-operaatioiden tukemiseksi.

5.1.2 OpenGL

OpenGL on Khronos Groupin kehittämä ohjelmointirajapinta. (The Khronos Group Inc, 2020) Se tarjoaa grafiikkalaitteistosta riippumattoman grafiikkarajapinnan ja on yksi ensimmäisistä grafiikka rajapinnoista. Node JS -ympäristössä OpenGL:lle ei ole olemassa aktiivisesti kehityksessä olevaa rajapintaa. Tämä estää OpenGL:n valitsemisen toteutus tavaksi.

5.1.3 Vulkan

Vulkan on OpenGL:n tavoin Khronos Groupin kehittämä ohjelmointirajapinta. Se tarjoaa modernimman ja matalamman tason rajapinnan grafiikkalaitteille. (The Khronos Group Inc, 2020) Sen käyttäminen Node JS -ympäristössä on mahdollista NVK JavaScript kirjaston avulla. (maierfelix, 2020) Palvelimella piirtämisen toteuttaminen Vulkanin avulla vaatii uuden renderöintimoottorin kirjoittamisen ja on siten erittäin haastava toteuttaa. Lisäksi jatkokehittäminen olisi vaikeampaa sillä kaikki uudet ominaisuudet, jotka tarvitsevat uusia piirtotapoja vaatisivat muutosten tekemisen kahteen eri renderöinti moottoriin.

5.1.4 Selenium

Selenium on selainautomaatioalusta. (Software Freedom Conservancy, 2020) Se mahdollistaa selain sovellusten testauksen automatisoinnin. Sen avulla olisi myös mahdollista suorittaa LightningChart JS -kirjaston selainversiota ja siten toteuttaa palvelimella renderöinti. Seleniumin käyttäminen on raskasta, sillä se käyttää selainta ja hyödyntää selaimissa olevia rajapintoja, joiden avulla on mahdollista hallita selaimen toimintaa. Palvelimella renderöinnin toteuttamiseksi pitäisi toteuttaa erillinen Node JS -alustaa varten oleva kirjasto, joka automatisoisi Node JS:n ja Seleniumin välisen kommunikoinnin siten, ettei LightningChart JS -kirjaston käyttäjän tarvitsisi huolehtia siitä.

5.1.5 Node JS yhteensopivuus tason avulla WebGL

Yksinkertaisin tapa ratkaista puuttuva WebGL tuki olisi käyttää rajapintaa, joka toteuttaa täyden WebGL -rajapinnan. Yksi tällainen projekti on headless-gl, se toteuttaa täyden WebGL 1.0.3 määritelmän. (stackgl, 2020) Tämä lähestymistapa ei vaadi muuta kuin yhteensopivuus rajapinnan asentamisen ja käyttämisen.

5.1.6 JSDOM

JSDOM on JavaScript kirjasto, joka implementoi monia verkkostandardeja Node JS -käyttöä varten. Se mahdollistaa virtuaalisen dokumenttimallin luomisen ja siten mahdollistaa samanlaisen dokumentin muokkaamisen kuin jos dokumentti olisi ladattuna selaimeen. (jsdom, 2020) LightningChart JS -käyttää monia dokumenttimallin funktiota, joten JSDOM on hyödyllinen LightningChart JS:n ajamiseen Node JS -alustalla.

JSDOM on MIT lisensoitu ja se on avoimen lähdekoodinkirjasto, joten sen käyttäminen on sallittua kaupallista käyttöä varten. (jsdom, 2020) Jos kirjaston kehittäminen joskus loppuu, on mahdollista ladata lähdekoodi ja jatkaa kehittämistä yrityksen sisällä.

5.2 Valittu tekniikka

JSDOM:ia päädyttiin käyttämään Node JS -alustalta puuttuvien ohjelmointirajapintojen korvaamiseen. Se ei kuitenkaan tue WebGL ohjelmointirajapintaa, joten WebGL tuki toteutetaan headless-gl kirjaston avulla. se mahdollistaa yksinkertaisimman ja helpoimman ylläpidettävän tavan toteuttaa palvelimella piirtäminen. JSDOM ja headless-gl eivät kuitenkaan riitä kaikkien vaadittujen rajapintojen tuomiseen Node JS -alustalle, vaan vielä lisäksi tarvitaan Canvas -rajapinta. Canvas rajapinta saadaan node-canvas nimisen paketin kautta. JSDOM on yhteensopiva node-canvas paketin kanssa, joten näiden kahden kirjaston yhdistäminen on yksinkertaista.

6 TOTEUTUS

6.1 Puuttuvien rajapintojen korvaus

Yksinkertaisin mahdollinen ohjelma, joka käyttää LightningChart JS -kirjastoa on yhden rivin JavaScript ohjelma.

```
const lcjs = require('@arction/lcjs')
```

Kuva 2. Yksinkertaisin mahdollinen ohjelma käyttäen LightningChart JS kirjastoa.

Tämä ohjelma ajettaessa Node JS prosessi kaatuu. Kaatuminen tapahtuu koska LightningChart JS -kirjaston sisällyttäminen ajaa pienen määrän koodia, joka käyttää rajapintoja, jotka eivät ole käytettävissä Node JS -alustalla. JSDOM ja node-canvas kirjastot toteuttavat nämä puuttuvat rajapinnat, joten ne lisäämällä ja pienillä muutoksilla LightningChart JS -kirjastoon, saadaan pienin mahdollinen ohjelma suoritettua kaatumatta.

6.2 Renderöinti

Tähän asti käytetty ohjelma ei tee vielä mitään hyödyllistä. Tavoitteena on saada LightningChart JS -kirjastolla luotua kuvaajia ja saada ne tallennettua kuvina. Tämä vaatii uuden ominaisuuden lisäämisen LightningChart JS -kirjastoon. Tämä ominaisuus on yhden kuvan renderöinti ja tämän renderöidyn kuvan saaminen sovelluskoodin puolelle. Tämän toteuttamiseksi LightningChart JS -kirjastoon lisättiin renderFrame -metodi. Tämä metodi mahdollistaa yhden kuvan renderöinnin tietyllä resoluutiolla.

```
const chart = lcjs.lightningChart().ChartXY()
chart.engine.renderFrame(1280, 720)
```

Kuva 3. Pienin mahdollinen ohjelma, joka renderöi kuvaajan.

Tämä ohjelma ei kuitenkaan vielä anna minkäänlaista kuvaajaa ulos. WebGL tuki puuttuu yhä. Headless-gl kirjasto toteuttaa tämän tuen, mutta sen sisällyttäminen ei ole yksinään riittävää. LightningChart JS ei pysty käyttämään headless-gl pakettia suoraan. LightningChart JS -hakee WebGL kontekstin Canvas -elementin kautta käyttämällä getContext -metodia.

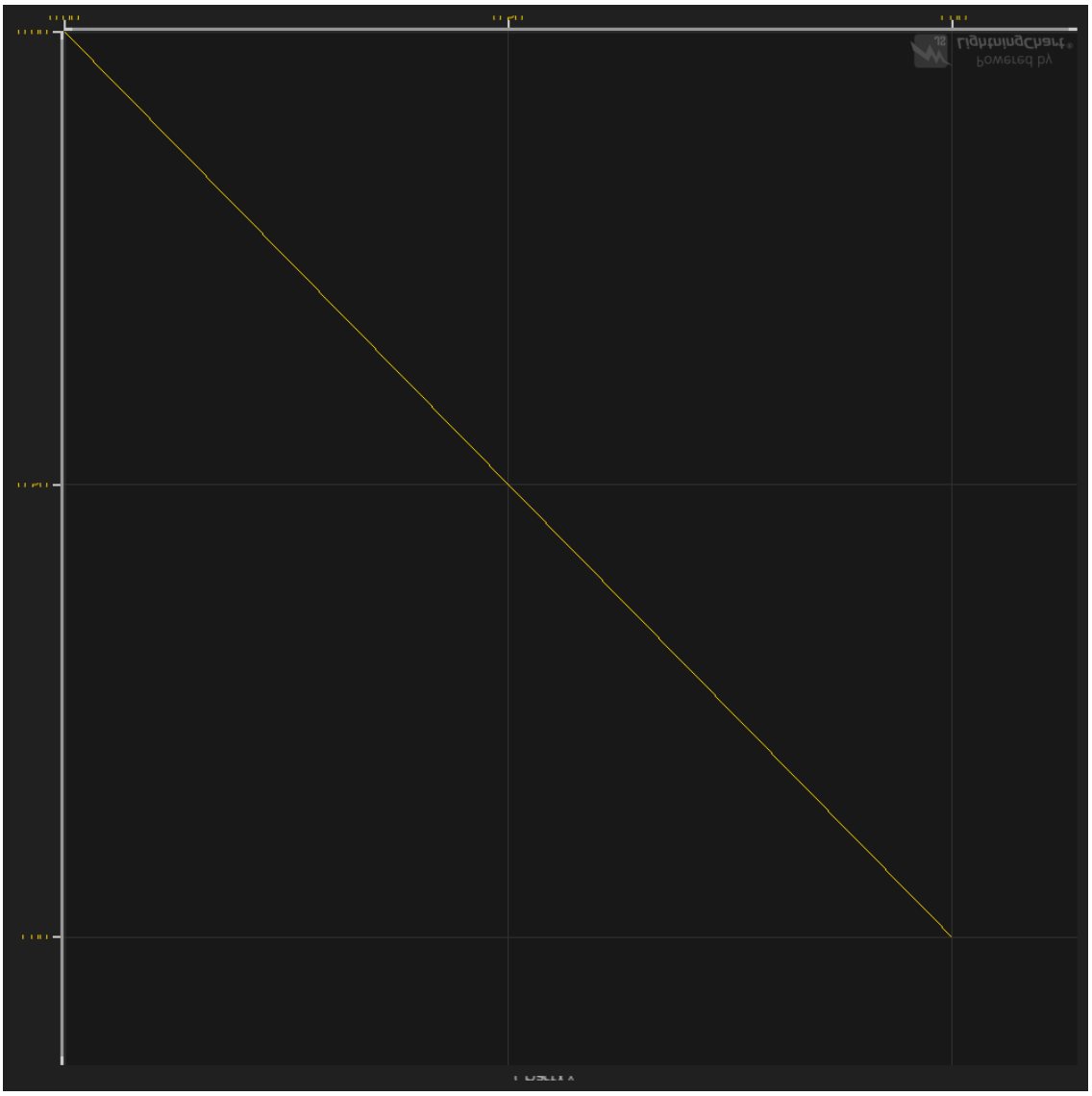
Canvas -elementin getContext -metodi voidaan ylikirjoittaa toisella funktiolla, joka pystyy tarjoamaan headless-gl:n tarjoaman kontekstin. Tätä ylikirjoittamista kutsutaan ”monkey patching”:iksi. (De Rosa, 2016)

```
import createContext from 'gl'
const orig_getContext = dom.window.HTMLCanvasElement.prototype.getContext
dom.window.HTMLCanvasElement.prototype.getContext = function () {
    if (arguments[0] === 'webgl') {
        const ctx = createContext(1, 1, arguments[1])
        ctx.resize = ctx.getExtension('STACKGL_resize_drawingbuffer').resize
        return ctx
    } else {
        return orig_getContext.apply(this, arguments)
    }
}
```

Kuva 4. Canvas -elementin getContext -metodin ylikirjoitus.

Uudessa getContext -metodissa WebGL kontekstia haettaessa metodia palauttaa headless-gl kirjaston tarjoaman WebGL kontekstin. Headless-gl tarjoaa myös mahdollisuuden muuttaa piirtopuskurin kokoa STACKGL_resize_drawingbuffer -laajennoksen kautta, tämä toiminnallisuus mahdollistaa usean eri resoluutioisen kuvan piirtämisen ilman että käytössä oleva kaavio tarvitsee poistaa ja luoda uudestaan. (stackgl, 2020) Normaalissa WebGL kontektissa tätä tukea ei ole, joten laajennos lisätään kontekstiin LightingChart JS:n käytettäväksi. Jos getContext -metodia kutsutaan jonkun muun kontekstin saamiseksi kuin WebGL, niin kutsutaan alkuperäistä getContext -metodia ilman muokkauksia. Näillä muokkauksilla saadaan ensimmäistä kertaa kuvaaja renderöityä Node JS -alustalla.

6.3 Renderöintiongelmat



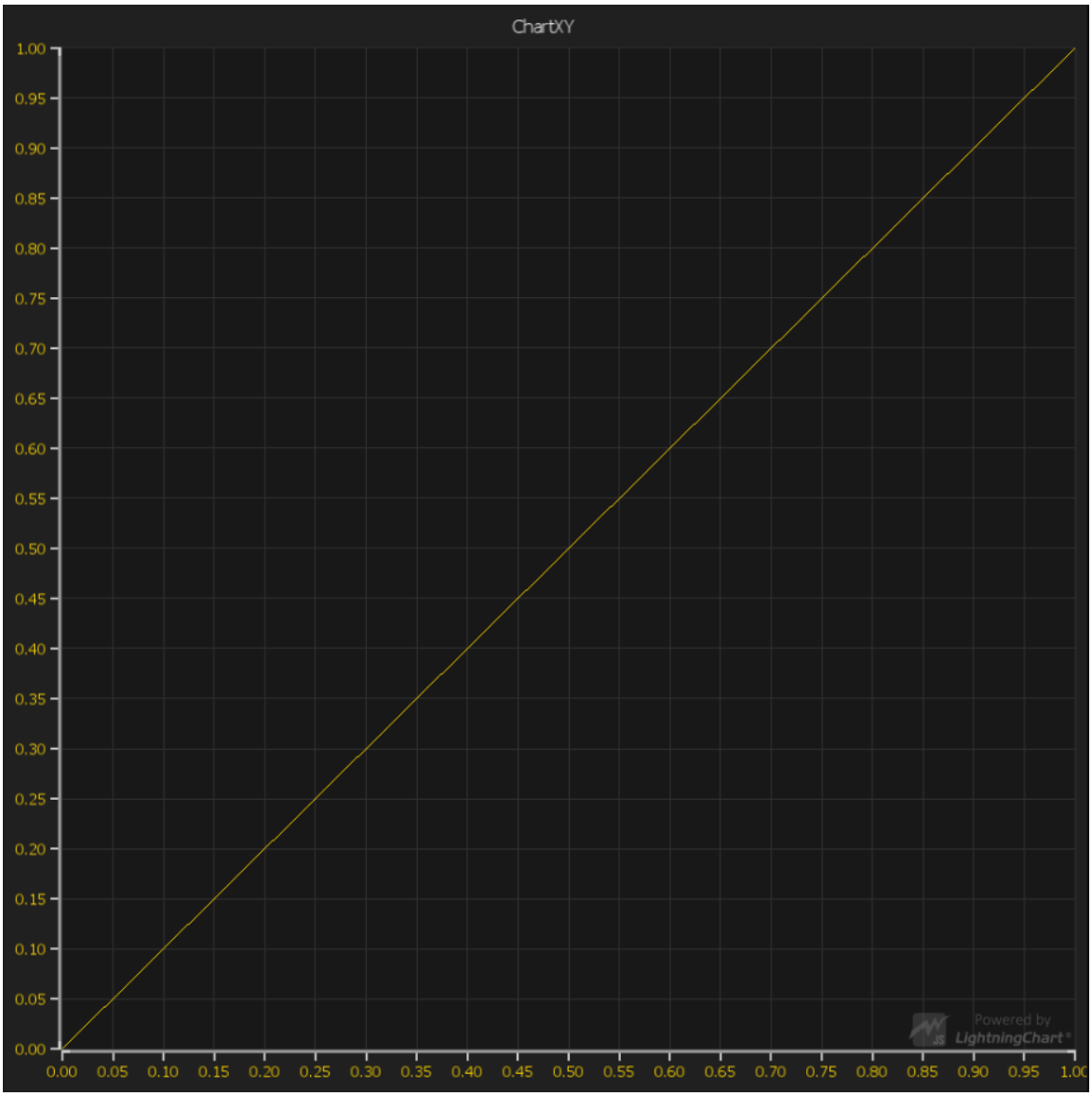
Kuva 5. Ensimmäinen kuvaaja Node JS alustalla renderöitynä.

Kuvasta huomataan heti muutama ongelma. Kaikki kuvassa olevat tekstit on piirretty vain pieniltä osilta. Lisäksi kuva on ylösalaisin.

Kuvan piirtyminen ylösalaisin johtuu siitä, miten WebGL hallitsee piirtopuskurin koordinaatit. Kuvan nollapiste ((0,0) koordinaatti) on vasemmassa yläkulmassa. Piirtopuskurin lukemiseen käytettävä readPixels -metodi palauttaa piirtopuskurin muokkaamatta sitä. Jotta kuvaaja saadaan piirrettyä oikein päin pitää readPixels -metodin palauttamaa puskuria muokata siten, että kuvan Y-koordinaatti saadaan käännettyä. Käytännössä tämä tarkoittaa kuvan pikselirivijärjestyksen vaihtamisen siten että tällä hetkellä alin rivi siirtyy ylimmäksi ja toisiksi alin toiseksi ylimmäksi.

Tekstin osittain piirtyminen johtuu LightningChart JS kirjaston tavasta mitata tekstin korkeus. Normaalisti teksti piirretään ensin Canvas -rajapinnalla toiseen, näkymättömään, canvas -elementtiin. Piirretyn tekstin korkeus mitataan asettamalla teksti näkymättömään span -elementtiin. Dokumentti

-rajapinta tarjoaa funktion tekstin korkeuden mittaamisen. Tekstin leveys on mahdollista mitata Canvas -rajapinnan metodilla. Näillä mitoilla piirretty teksti siirretään tekstuuriin ja piirretään näkyväksi WebGL:n avulla. Node JS ympäristössä nämä rajapinnat eivät toimi täysin samalla tavalla. Span -elementti ei muokkaa kokoaan automaattisesti pitääkseen tekstin näkyvissä. Tämän takia tekstin korkeus on vain LightningChart JS:n lisäämän marginaalin korkuinen. Tekstiä piirrettäessä kuitenkin tiedetään haluttu fonttikoko, tämän tiedon avulla voidaan asettaa tekstin korkeudelle alaraja. Tällöin jotkut erikoismerkit voivat tulla hieman leikatuiksi, mutta suurin osa tekstistä piirtyy oikein.



Kuva 6. Suurimmat renderöintiongelmat korjattuna.

6.4 Animaatiot

LightningChart JS -animoi normaalisti kaiken datan lisäämisen. Palvelimella piirrettäessä animaatiot aiheuttavat ongelman. Koska tavoitteena ei ole pysty renderöimään videota kaaviosta animaatioiden kanssa vaan yksi kuva mahdollisimman nopeasti, animaatiot olisi hyvä pystyä ottamaan pois päältä vakiona. Tämä toteutettiin lisäämällä kaikkiin LightningChart JS -kirjaston osiin, jotka on mahdollista animoida, metodi, jolla animaatiot saa poistettua käytöstä suorituksen aikana tai jo ennen suoritusta.

6.5 Palvelinympäristössä käyttö

Koska tavoitteena on luoda renderöintituki palvelin käyttöön, tulee käytön olla mahdollista palvelinympäristössä. Windows -pohjaisilla palvelimilla käyttäminen onnistuu samalla tavalla kuin normaalissa työpöytäympäristössä. Linux -palvelimilla käyttäminen ei aina ole aivan yhtä yksinkertaista.

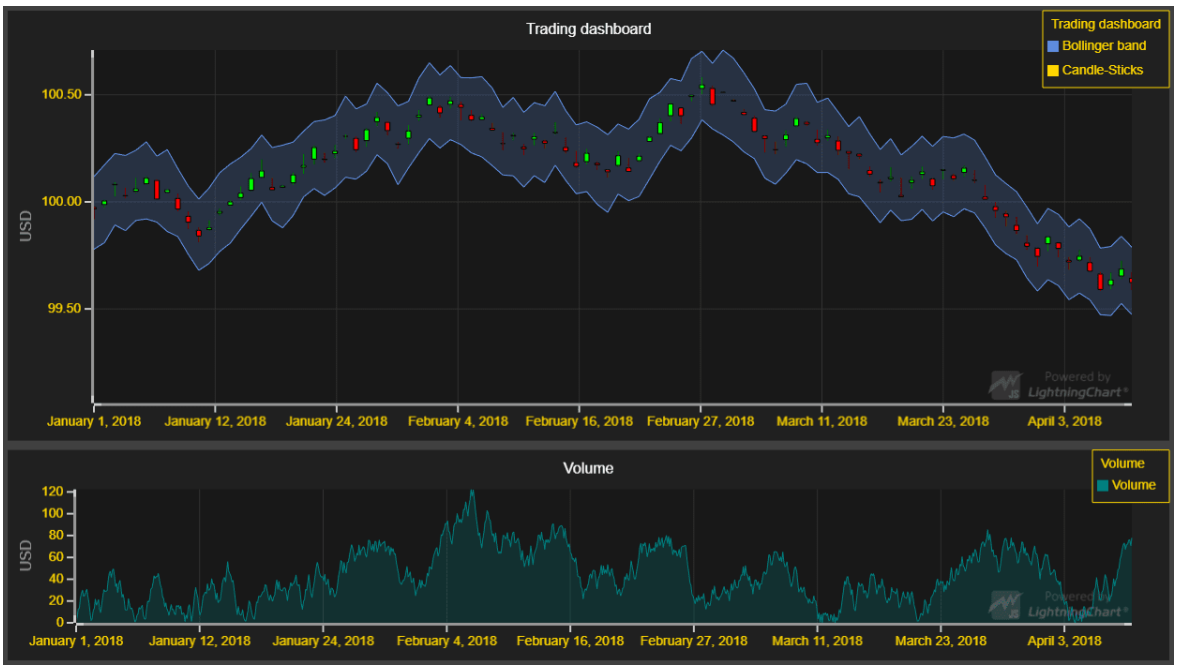
Linux -palvelin ympäristö on yleensä ”headless”-palvelin eli palvelimella ei ole lainkaan graafista käyttöliittymää. Tämä aiheuttaa ongelman, LightningChart JS:n renderöinti vaatii graafisenjärjestelmän. Onneksi Linux -ympäristössä useasti käytettävään X.Org Server näyttöpalvelimeen on olemassa X-palvelin, joka mahdollistaa virtuaalisen näyttöpuskurin luonnin ja käyttämisen ohjelman suorittamiseen, Xvfb. (Wiggins, 2010) Tämän X-palvelimen käyttäminen mahdollistaa LightningChart JS:n käytön Linux palvelimilla ilman graafista käyttöliittymää.

Xvfb:tä käytetään antamalla halutut näyttöparametrit ja haluttu ohjelma komentoriviparametreinä ”xvfb-run” nimiselle ohjelmalle.

```
xvfb-run -s "-ac -screen 0 1280x720x24" <node program>
```

Kuva 7. Node JS sovelluksen ajo xvfb X-palvelimen kanssa.

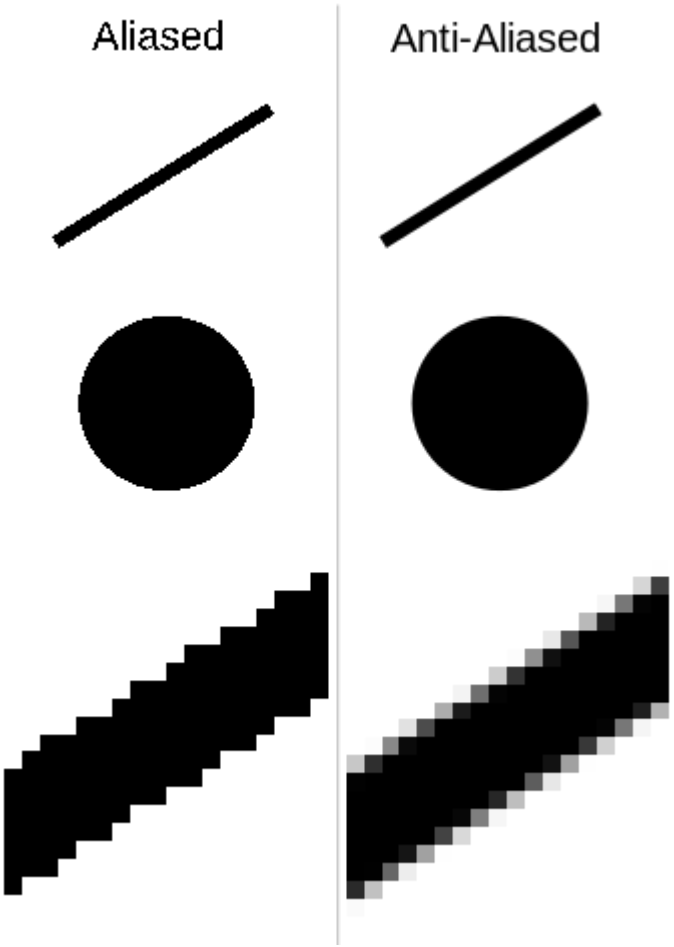
Virtuaalinen näyttöpuskuri ei kuitenkaan ole usein yksinään riittävä muutos, jotta palvelimella voidaan renderöidä. Erillinen grafiikkalaittesto puuttuu suuresta osasta Linux -pohjaisia palvelimia. Koska WebGL on pääsääntöisesti laitteistokiihdytetty renderöintitapa, tarvitsee se erillisen grafiikkalaitteiston. Onneksi grafiikkalaitteiston puutuminen ja grafiikkakirjastojen käyttö on yleistä ja tämä ongelman on ratkaistu tekemällä ohjelmistopohjainen renderöinti. Jotta ohjelmistopohjaista renderöintiä voi käyttää tulee kohdelaitteelle olla asennettuna Mesa -pohjaiset laitteistoajurit. Mesa on avoimenlähdekoodin implementaatio OpenGL:stä. (Paul, 2020)



Kuva 8. Linux palvelimella renderöity kaavio.

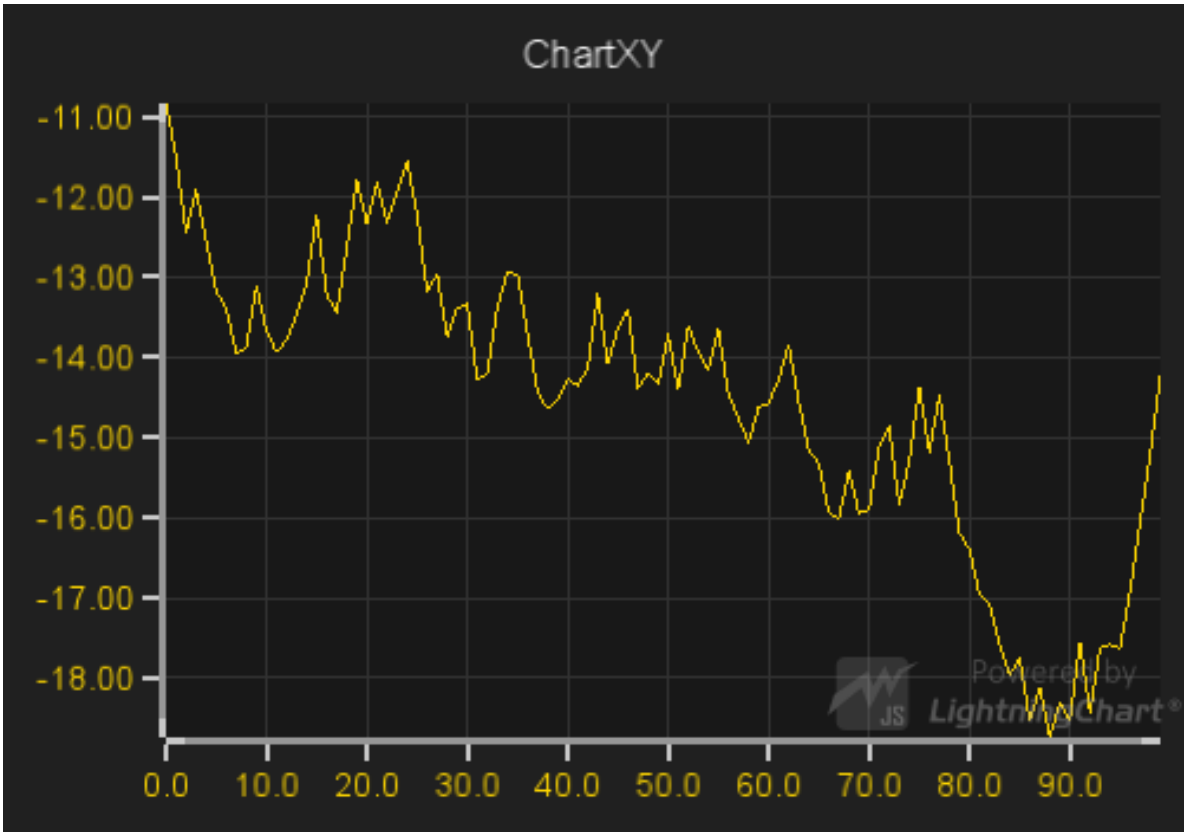
6.6 Antialiasointi

WebGL määritys ei vaadi, että implementaatio toteuttaa antialiasointituen. (The Khronos Group Inc, 2014) Headless-gl paketin implementaatio WebGL määrityksestä ei toteuta antialisointia. Antialisoinnin puuttuminen aiheuttaa aliasointia eli viivan kulmikkuutta.

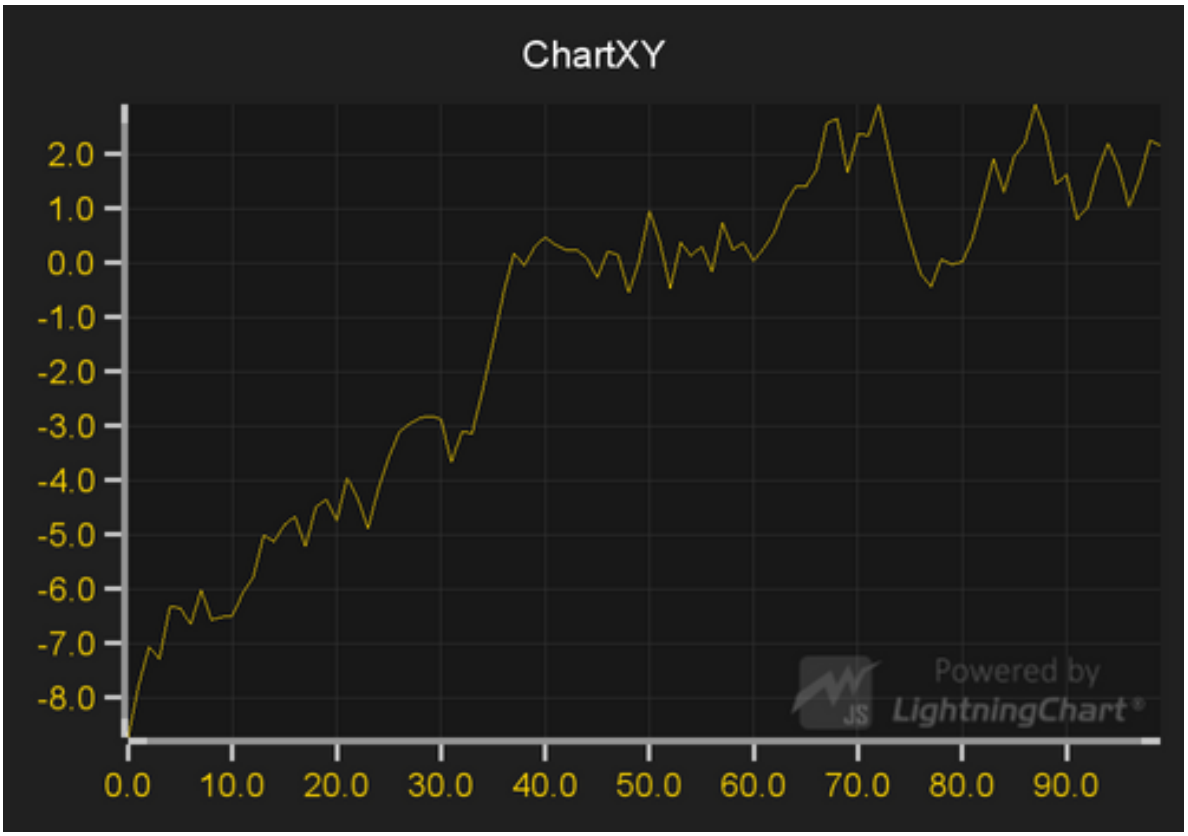


Kuva 9. Aliasointi ja antialiasointi. (Wikimedia Commons contributors, 2016)

Antialisoinnin puuttumista pyrittiin korvaamaan tekemällä mahdollisuus renderöidä haluttu kaavio suuremmassa koossa ja pienentämällä tämä kuva haluttuun kokoon. Tällä tavalla kuvan aliasointia pystytään rajoittamaan hieman mutta ei kuitenkaan yhtä hyvin kuin jos varsinainen antialisointi implementaatio olisi käytettävissä.



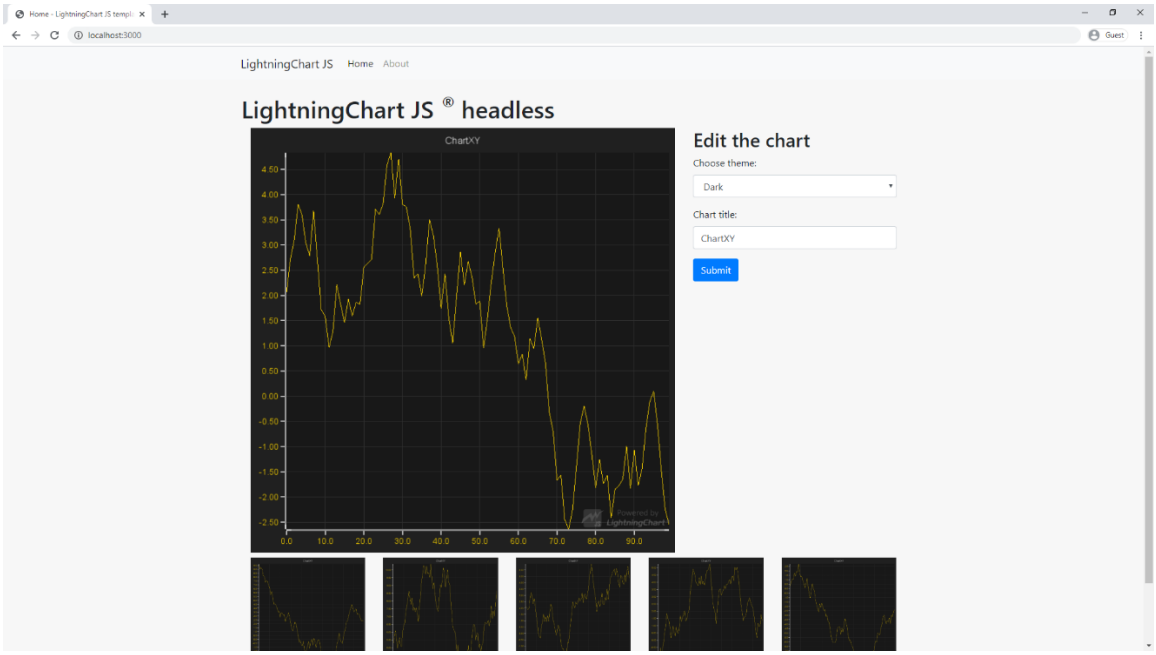
Kuva 10. Kuvaaja ilman antialisointia.



Kuva 11. Antialisoitu kuvaaja.

Antialisoidussa kuvaajassa (Kuva 11), viiva on pehmeämpi ja näyttää enemmän viivalta kuin kuvaajassa (Kuva 10) jossa, ei ole minkään laista antialisointia. Ilman antialisointia viivasta tulee kulmikas.

6.7 Esimerkkipalvelin



Kuva 12. Kuvakaappaus LightningChart JS -esimerkkipalvelimen käyttöliittymästä.

Kun toteutus oli valmis, tehtiin sen käytöstä esimerkki, joka julkaistaan avoimen lähdekoodin sovelluksena, jotta asiakkaat voivat nähdä toimivan koodin ja ryhtyä kasaamaan omia sovelluksia tämän koodin pohjalta. Tämä pohja julkaistiin julkisena avoimenlähdekoodin projektina GitHubiin Arctionin organisaation alle lcjs-node-template nimellä.

Esimerkkipalvelin on Express pohjainen Node JS -palvelin, joka näyttää kuusi eri kuvaajaa satunnaisella datalla. Express on yksi yleisimmistä palvelin runkokirjastoista. Se mahdollistaa Node JS:n käytön palvelimien kehittämiseen yksinkertaisesti ja helposti. (OpenJS Foundation, 2020) Esimerkkipalvelin on tarkoitettu projektipohjaksi, joten itse palvelin on kehitetty mahdollisimman yksinkertaiseksi.

7 YHTEENVETO

Työssä tutkittiin useita eri tekniikoita työn toteuttamiseksi. Näistä eri vaihtoehdoista valittiin paras yhdistelmä. Lopullisessa toteutuksessa käytettiin JSDOM, node-canvas ja headless-gl nimisiä kirjastoja tarvittavien ohjelmointirajapintojen tuomiseksi Node JS -alustalle. Näiden kirjastojen avulla oli mahdollista kehittää toinen JavaScript kirjasto, minkä avulla LightningChart JS -kirjastoa on mahdollista käyttää Node JS -alustalla. Tämä kirjasto julkaistiin npm -rekisteriin ja GitHubiin @arction/lcjs-headless nimisenä avoimen lähdekoodin kirjastona, 28. huhtikuuta 2020. Kirjasto julkaistiin MIT-ohjelmistolisenssin alla. Kirjasto mahdollistaa kenen tahansa käyttää LightningChart JS kirjastoa Node JS -ympäristössä ilmaista LightningChart JS yhteisölisenssiä käyttämällä.

Työ oli haastava ja sen aikana ilmeni useita ennalta arvaamattomia ongelmia. Suurin osa näistä ongelmista oli kuitenkin mahdollista ratkaista, aivan kaikkia ongelmia ei työn aikana pystytty ratkaisemaan ja nämä ongelmat on kirjattu ylös Arctionin kehitysjärjestelmiin ja tullaan ratkaisemaan myöhemmässä vaiheessa. Ongelmista huolimatta työ onnistui hyvin ja työn tavoitteisiin päästiin.

Jatkokehityksenä voisi toteuttaa animoidun renderöinnin videoksi. Tällä tavalla pystyisi luomaan kuvaajia, joissa esimerkiksi aika esitetään videon muodossa muutoksina eikä yhtenä akselina kuvaajassa.

8 LÄHDELUETTELO

Arction Oy. (2020). *Javascript High Performance Charts | WebGL Charts Library*. Haettu 10. Toukokuuta 2020 osoitteesta https://www.arction.com/lightningchart-js/

Automattic. (Huhtikuu 2020). *Automattic/node-canvas: Node canvas is a Cairo backed Canvas implementation for NodeJS.* Haettu 28. Huhtikuuta 2020 osoitteesta https://github.com/Automattic/node-canvas

De Rosa, A. (05. Joulukuu 2016). *Monkey patching in JavaScript - Aurelio De Rosa blog*. Haettu 1. Toukokuuta 2020 osoitteesta https://www.audero.it/blog/2016/12/05/monkey-patching-javascript/

Google. (Huhtikuu 2020). *ANGLE - Almost Native Graphics Layer Engine*. Haettu 28. Huhtikuuta 2020 osoitteesta https://chromium.googlesource.com/angle/angle/+/master/README.md

jsdom. (Huhtikuu 2020). *jsdom/jsdom: A JavaScript implementation of various web standards, for use with Node.js*. Haettu 28. Huhtikuuta 2020 osoitteesta https://github.com/jsdom/jsdom

maierfelix. (Huhtikuu 2020). *NVK*. Haettu 28. Huhtikuuta 2020 osoitteesta https://maierfelix.github.io/nvk/1.1.126/

MDN Web Docs. (Huhtikuu 2020). *Canvas API - Web APIs | MDN*. Haettu 28. Huhtikuuta 2020 osoitteesta https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API

MDN Web Docs. (Huhtikuu 2020). *DOM (Document Object Model) - MDN Web Docs Glossary: Definitions of Web-related terms | MDN*. Haettu 28. Huhtikuuta 2020 osoitteesta https://developer.mozilla.org/en-US/docs/Glossary/DOM

MDN Web Docs. (Maaliskuu 2020). *JavaScript | MDN*. Haettu 28. Huhtikuuta 2020 osoitteesta https://developer.mozilla.org/en-US/docs/Web/JavaScript

MDN Web Docs. (Maaliskuu 2020). *WebGL: 2D and 3D graphics for the web - Web APIs | MDN*. Haettu 4. Toukokuuta 2020 osoitteesta https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API

MDN Web Docs. (Huhtikuu 2020). *WebGL2RenderingContext - Web APIs | MDN*. Haettu 5. Toukokuuta 2020 osoitteesta https://developer.mozilla.org/en-US/docs/Web/API/WebGL2RenderingContext

MDN Web Docs. (Huhtikuu 2020). *Window - Web APIs | MDN*. Haettu 5. Toukokuuta 2020 osoitteesta https://developer.mozilla.org/en-US/docs/Web/API/Window

Microsoft. (Maaliskuu 2020). *TypeScript - JavaScript that scales.* Haettu 4. Toukokuuta 2020 osoitteesta https://www.typescriptlang.org/

OpenJS Foundation. (Maaliskuu 2020). *Node.js*. Haettu 28. Huhtikuuta 2020 osoitteesta https://nodejs.org/en/

OpenJS Foundation. (Huhtikuu 2020). *Node.js v12.16.2 Documentation*. Haettu 28. Huhtikuuta 2020 osoitteesta https://nodejs.org/dist/latest-v12.x/docs/api/index.html

Paul, B. (Huhtikuu 2020). *Mesa 3D Graphics Library*. Haettu 4. Toukokuuta 2020 osoitteesta https://www.mesa3d.org/intro.html

Software Freedom Conservancy. (Huhtikuu 2020). *SeleniumHQ Browser Automation*. Haettu 28. Huhtikuuta 2020 osoitteesta https://www.selenium.dev/

stackgl. (Huhtikuu 2020). *stackgl/headless-gl: Windowless WebGL for node.js*. Haettu 4. Toukokuuta 2020 osoitteesta https://github.com/stackgl/headless-gl

The Khronos Group Inc. (27. Lokakuu 2014). *WebGL Specification*. Haettu 10. Toukokuu 2020 osoitteesta https://www.khronos.org/registry/webgl/specs/1.0/

The Khronos Group Inc. (Huhtikuu 2020). *OpenGL*. Haettu 4. Toukokuuta 2020 osoitteesta https://www.opengl.org/

The Khronos Group Inc. (Huhtikuu 2020). *Vulkan Overview - The Khronos Group Inc*. Haettu 5. Toukokuuta 2020 osoitteesta https://www.khronos.org/vulkan/

Wiggins, D. (Joulukuu 2010). *XVFB*. Haettu 5. Toukokuuta 2020 osoitteesta https://www.x.org/releases/X11R7.6/doc/man/man1/Xvfb.1.xhtml

Wikimedia Commons contributors. (24. Elokuu 2016). File:Anti-aliasing demo.svg. Wikimedia Commons, the free media repository. Haettu 10. Toukokuuta 2020 osoitteesta https://commons.wikimedia.org/w/index.php?title=File:Anti-aliasing_demo.svg&oldid=204715338