Miika Henttonen

Teksti-TV-editorin kehittäminen moderneilla web-teknologioilla

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Insinöörityö

16.2.2016

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Miika Henttonen Teksti-TV-editorin kehittäminen moderneilla web-teknologi-oilla 32 sivua 16.2.2016
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja(t)	Lehtori Ilpo Kuivanen Palvelujohtaja Jan Enlund

Insinöörityön tarkoituksena oli toteuttaa nykyaikainen selainpohjainen versio Teksti-TV-editorista. Sovelluksella oli tarkoitus korvata vanha Java-pohjainen käyttöliittymä. Vanhan Teksti-TV-editorin ominaisuudet kopioitiin uuteen versioon siten, että käytettävyys pysyi samana. Joiltakin osin editorin käytettävyyttä kehitettiin paremmaksi. Tavoitteena oli, että uusi editori mukautuisi näytön ruudun koon mukaan ja toimisi myös mobiililaitteilla.

Työssä esiteltiin käytetyt Javascript-kirjastot ja menetelmät. Osuudessa selvitettiin Bootstrap-, Backbone- ja RequireJS-kirjastojen käyttäminen osana selainpohjaista sovellusta. Tämän lisäksi osuudessa esiteltiin npm- ja bower-paketinhallintajärjestelmät sekä HTML5 Canvasin toiminta ja käyttäminen. Myös sovelluksen rakenne yleisellä tasolla esiteltiin. Käytettyjen teknologioiden esittelyn ohessa myös kerrottiin, miksi kyseiset teknologiat valittiin osaksi insinöörityön toteutusta. Osuuden lopuksi esiteltiin projektin kehitysversion kokoaminen tuotantoversioksi RequireJS- ja Gulp-kirjastoja käyttäen.

Työn lopuksi käytiin laajasti läpi Teksti-TV-editorin toteutuksessa esiin tulleita asioita ja ongelmia. Editorin käyttöliittymän responsiivisuus Bootstrap-kirjastoa käyttäen, skaalautuminen sekä mukautuminen kosketusnäyttöön esiteltiin. Osuudessa esiteltiin myös merkkitoimintoja ja merkkien vierittämistä ja rivittämistä.

Avainsanat	Teksti-TV, Javascript, Bootstrap, HTML5 Canvas, npm

Author(s) Title Number of Pages Date	Miika Henttonen Developing Teletext Editor with Modern Web Tools 32 pages 16 February 2016
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Ilpo Kuivanen, Senior Lecturer Jan Enlund, Senior Service Manager

The purpose of this thesis was to develop a modern web-based version of a Teletext editor. The new editor was meant to replace a Java-based legacy user interface. The features of the legacy editor were copied to the new version, so that the old user experience could be maintained as closely as possible to the original. In some cases the editor's functionality was improved. One objective was that the editor would be responsive and would thus also work with touch devices.

The usage of the Bootstrap, Backbone and RequireJS Javascript libraries in web-based development is detailed in the study. The npm and bower package managers are explained, as is the usage and functioning of the HTML5 Canvas. The reasons why the aforementioned technologies were chosen and the structure of the application are also explained. The study also details how a development version of the application is built into a package ready for production.

Finally, this thesis reviews many of the issues and problems encountered during development. The editor's responsiveness, scaling and touch features are explained, as are the various character functions and the character and line wrap features.

Keywords	Teletext, Javascript, Bootstrap, HTML5 Canvas, npm

Sisällys

Lyhenteet

CSS	Cascading Style Sheets. Selaimien ymmärtämä nettisivujen tyyliohje.
DPR	Device Pixel Ratio. Käytettyjen pikselien määrä verrattuna näytön resoluutioon.
HTML5	Hypertext Markup Language. Viides versio nettisivujen koodaamiseen käytetystä hypertekstin kuvauskielestä.
JSON	Javascript Object Notation. Tiedonsiirtoon tarkoitettu tiedostomuoto, jossa data on esitetty avain-arvo-pareina.
NPM	Node Package Manager. Node.js-ajoympäristön paketinhallintajärjestelmä.
REST	Representational State Transfer. Arkkitehtuurimalli ohjelmointirajapintojen toteuttamiseen.
SEMVER	Semantic Versioning. Versionumeroinnin spesifikaatio.
SVG	Scalable Vector Graphics. Kaksiulotteisten vektorikuvien kuvauskieli.
WYSIWYG	What You See Is What You Get. Ohjelmisto, jossa muokattava sisältö näyttää samalta kuin lopputulos.

1 Johdanto

Tässä insinöörityössä toteutetaan nykyaikainen selainpohjainen versio Teksti-TV-editorista, jolla luodaan Teksti-TV-sivuja televisiolähetykseen. Sovelluksella on tarkoitus korvata vanha Java-pohjainen käyttöliittymä. Työ tehtiin FNX Solutions Oy:lle, joka on Helsingissä toimiva tietotekniikkaratkaisuja kehittävä 20 työntekijän yritys.

Vanha Teksti-TV-editori on toteutettu Java-applettina. Editori on kaikinpuolin toimiva, mutta ikävä puoli siinä on nimenomaan se, että käyttäjän pitää asentaa Java. HTML5:n mukana on kuitenkin tullut uusi Canvas-niminen elementti, jolla pitäisi olla mahdollista korvata vanha Java-appletti.

Työn tarkoituksena on kopioida vanhan editorin toiminnallisuus selainpohjaiseen ohjelmaan, jonka palvelin voi tarjoilla käyttäjille. Toiminnallisuuksia ovat mm. tekstin luominen, poisto, leikkaus ja kopiointi, merkkien ja taustojen värien asettaminen, tekstin vierittäminen ja rivittäminen sekä alasivujen luonti, poisto, kopiointi ja numerointi. Editorin WYSIWYG-periaate (What You See Is What You Get) on säilytettävä.

Tarkoituksena on tehdä uudesta editorista mahdollisimman samannäköinen suosituimpien selaimien kesken. Editorin on tarkoitus toimia niin pöytäkoneilla kuin mobiililaitteilla. Tämä tarkoittaa sitä, että editorin pitää skaalautua tarpeen mukaan pienille ja isoille ruuduille, mikä tarkoittaa sitä, että editorin sisältämän HTML-sivun pitää olla responsiivinen. Editoria pitää voida käyttää näppäimistöllä ja hiirellä, sekä kosketusnäytöllä.

Omana tavoitteenani on projektin kautta tutustua nykyaikaisiin web-teknologioihin ja tätä kautta tehdä editorista mahdollisimman modulaarinen ja ylläpidettävä kokonaisuus. Tarkoituksena on myös tehdä editorin kehitysympäristöstä yksinkertainen ja helposti ymmärrettävä, eli sellainen, että tarvittaessa editorin kehitystyökalut voitaisiin ladata ja editorin tuotantoversio voitaisiin koota muutamalla komentorivikomennolla.

Työssä selostetaan sovelluksen rakenne, käytettyjä menetelmiä ja teknologioita sekä tutustutaan editorin toteutuksessa esiintyneisiin asioihin ja ongelmiin.

2 Tausta

Teksti-TV syntyi Isossa-Britanniassa 70-luvun loppupuolella, ja Suomeen se rantautui vuonna 1981 [1]. Se hyödynsi televisiovastaanottimiin lähetettävän analogisen signaalin välttämätöntä ns. tyhjää tilaa, joka liittyy kuvaputkinäytön toimintaperiaatteeseen. Tähän tyhjään tilaan voitiin kuitenkin lähettää Teksti-TV:n sisältö digitaalisessa muodossa [2]. Teksti-TV sisältää mm. uutisia, urheilutuloksia, TV-kanavien ohjelmaoppaita, säätiedotteita sekä lentojen, kaukojunien ja bussien aikatauluja.

Teksti-TV koostuu sadan sivun alueista, joita kutsutaan makasiineiksi. Makasiineja voidaan välittää kerrallaan kahdeksan. Tästä syystä Teksti-TV käsittää sivut 100 – 899. Sivu koostuu 22 rivistä, joilla on jokaisella 40 merkkiä. Sivulla voi olla alasivuja, joita voisi standardin mukaan maksimissaan olla jopa 3200, mutta Suomessa alasivujen lukumäärä on rajoitettu 99:ään.

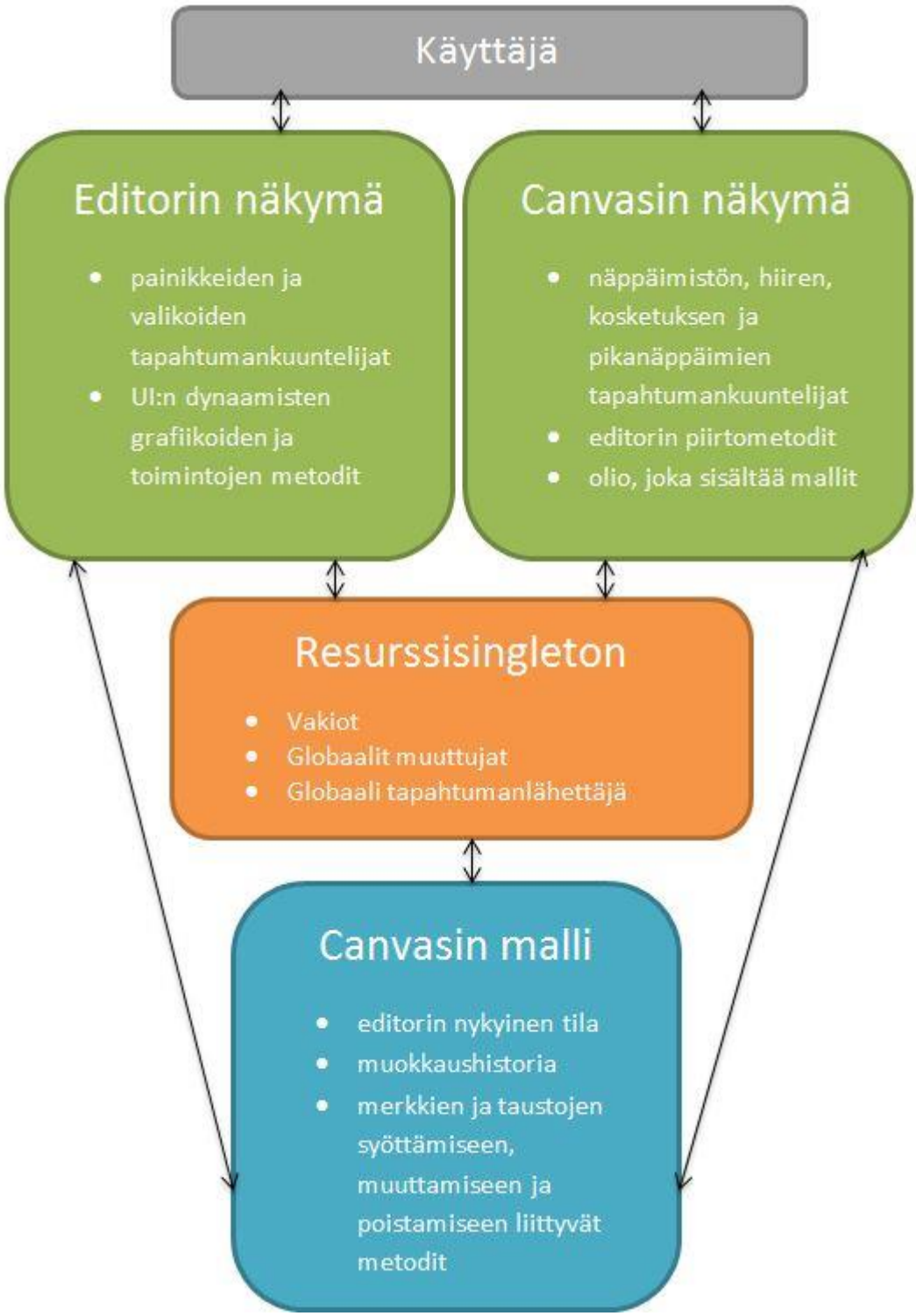
Iästään huolimatta Teksti-TV:llä on edelleenkin suuri suosio Suomessa ja muissa Pohjoismaissa. Taloututkimuksen mukaan Teksti-TV:llä on viikkotasolla 1,2 miljoonaa käyttäjää ja päivittäin 770 000. Eräänä syynä saattaa olla, että Teksti-TV on edelleenkin erittäin helppo tapa tarkistaa uusimmat uutisotsikot ja urheilutulokset. [3; 4.]

3 Rakenne

Tässä luvussa käsitellään yleisellä tasolla sovelluksen rakenne. Kuten kuvasta 1 voi nähdä, sovelluksessa on kaksi erilaista näkymää, joiden kautta käyttäjä vaikuttaa itse editorin tilaan ja esillä olevaan näkymään. Editorin yleinen näkymä käsittelee painikkeiden ja valikoiden painallukset ja kutsuu tarvittaessa Canvas-mallia ja resurssisingletonia. Canvasin näkymä taas sisältää piirtoalustaan liittyvät tapahtumankäsittelijät kuten näppäinten ja hiiren painallusten käsittelyt sekä pikanäppäimet. Tarvittaessa sekin kutsuu Canvas-mallia ja resurssisingletonia.

Resurssisingleton sisältää sovelluksen vakiot, globaalit muuttujat ja globaalin tapahtumanlähettäjän. Tämän ansiosta vakioita ei tarvitse määritellä useassa paikassa, ja singleton pitää huolen siitä, että tätä kyseessä olevaa oliota on olemassa kerrallaan vain yksi kappale. Kaikki muuttujat, jotka eivät ole yksittäiselle mallille uniikkeja, ovat resurssisingletonin sisällä.

Editorin leikepöytä on eräs globaaleista muuttujista. Kun käyttäjä kopioi tai leikkaa merkkejä jossakin alasivussa, leike voidaan liittää mihin tahansa muuhun alasivuun. Jokaisella alasivulla on oma muokkaushistoriansa.



Kuva 1. Havainnekuva sovelluksen rakenteesta.

Backbone on työhön valittu Javascript-ohjelmistokehys, jonka tarkoituksena on luoda web-sovelluksiin hyvä rakenne. Se perustuu malleihin, kokoelmiin ja näkymiin. Mallit yhdistetään JSON:in (Javascript Object Notation) avulla REST-rajapintaan (Representational State Transfer).

Globaali tapahtumanlähettäjä on Backbonen ominaisuus, joka mahdollistaa omien tapahtumien luomisen ja liittämisen näkymään tai malliin. Sovelluksessa tapahtumanlähettäjää käytetään laukaisemaan tapahtumia mallin ja näkymien sisällä. Malli ei tiedä näkymistä mitään, joten malli laukaisee tapahtuman ja sen omistava näkymä käsittelee sen. Näkymät eivät myöskään tiedä toisista näkymistä mitään, joten tapahtumanlähettäjän avulla ne pystyvät laukaisemaan tarvittaessa tapahtumia toisissa näkymissä. Näin esimerkiksi käyttäjän kopioidessa merkkejä leikepöydälle tapahtumanlähettäjä laukaisee editorin näkymän metodin, joka pitää huolen, että Liitä-painike aktivoidaan.

Canvasin näkymä pitää sisällään editorin alasivut eli mallit. Malli vaihdetaan molempiin näkymiin, kun käyttäjä vaihtaa alasivua. Sovelluksessa on siis kaksi näkymää ja enintään 99 alasivua.

4 Käytetyt teknologiat

Tässä luvussa käydään läpi työssä käytettyjä teknologioita, kuten Javascriptin paketinhallintajärjestelmiä, kirjastoja ja kokoamistyökaluja. Versionhallintana käytettiin Gitiä ja sen kanssa työjärjestyksenä hieman yksinkertaistettua Gitflow'ta [5; 6]. Koko projektin perustana toimi Node.js. Node.js on ajoympäristö, joka perustuu Googlen kehittämään V8 Javascript -moottoriin. Ajoympäristön avulla ajettiin projektin paketinhallintaa ja kasaustyökaluja.

4.1 Paketinhallinta

Paketinhallinta takaa, että kehittäjillä on käytössään yhteensopivat versiot käytetyistä kirjastoista. Npm on Node.js-ajoympäristön paketinhallintajärjestelmä. Se on suunnattu palvelinpuolelle, mutta sitä käytetään myös käyttöliittymäpuolella. Järjestelmä on käytännössä vain yksi JSON-tiedosto, joka pitää sisällään muiden hyödyllisten tietojen ohella ennen kaikkea listauksen projektin riippuvuuksista.

Riippuvuuksilla tarkoitetaan tässä projektissa käytettyjä kirjastoja. Riippuvuudet voidaan jakaa tuotantoon tai kehitykseen kuuluviksi, jolloin projektin voi halutessaan alustaa ilman kehityksessä käytettäviä kirjastoja.

Npm käyttää SEMVER-versionumerointia (Semantic Versioning). Tämän ansiosta käyttäjä voi määrittää, mihin asti käytössä olevia kirjastoja voi päivittää. Versionumero koostuu kolmesta eri numerosta (esim. 1.2.3). Ensimmäinen numero tarkoittaa täysin uutta versiota, joka ei ole yhteensopiva pienempien versionumeroiden kanssa. Toinen numero tarkoittaa osaversiota, joka lisää uusia toiminnallisuuksia, jotka ovat taaksepäin yhteensopivia. Viimeinen numero on varattu taaksepäin yhteensopivia korjauksia varten, jotka eivät lisää toiminnallisuuksia. [7.]

Sopiva versio voidaan ilmoittaa yhdistämällä versionumero operaattoreihin. Operaattorit edeltävät versionumeroa. Operaattoreina käytetään:

- < pienempi kuin
- <= pienempi tai yhtä suuri kuin
- > suurempi kuin
- >= suurempi tai yhtä suuri kuin, tai
- = yhtä suuri kuin.

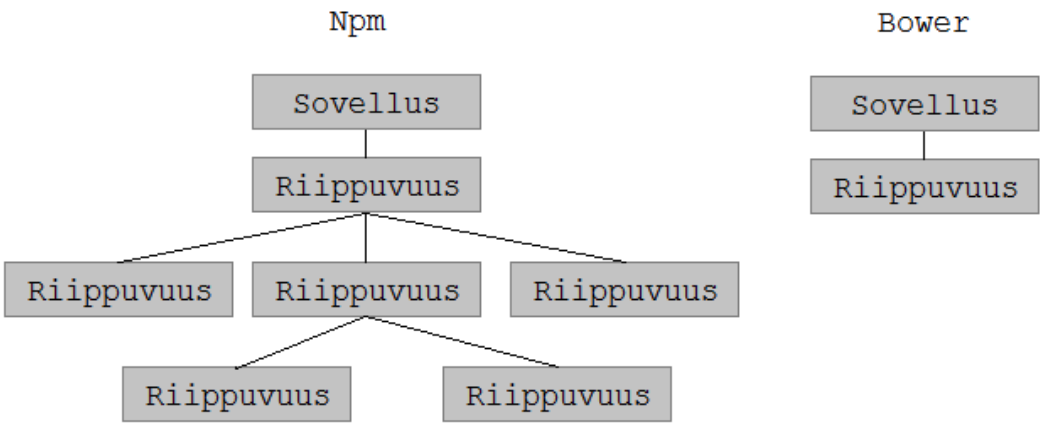
Yhtä suuri kuin -operaattori on oletusarvo eikä sitä ole pakko merkitä. Käytännössä npm asentaessaan paketteja kuitenkin lisää niiden eteen ~-merkin. Tällä operaattorilla sallitaan korjauksien asentaminen, jos toinen versionumero on merkitty (esim 1.2.x) ja osaversioiden asentaminen, jos toista versionumeroa ei ole merkitty (esim. 1.x). Toisin sanoen npm oletusarvoisesti sallii korjauksien asentamisen, mutta ei muita muutoksia. [8.]

Kun paketinhallintajärjestelmä on käytössä, kirjastojen asentaminen, päivittäminen, versioiden hallinta ja poistaminen onnistuvat hyvin yksinkertaisilla npm-komennoilla:

- npm install
- npm update
- npm uninstall.

Bower on käyttöliittymäpuolelle suunnattu paketinhallintajärjestelmä [9]. Se on hyvin samankaltainen kuin npm. Se käyttää samaa versionumerointia, ja riippuvuudet listataan yhdessä JSON-tiedostossa ja komennotkin ovat lähes samat.

Npm:n riippuvuusjärjestelmä on sisäkkäinen, kun taas Bowerin on täysin tasainen, (katso kuva 2).



Kuva 2. Npm:n ja Bowerin riippuvuusjärjestelmien ero [10].

Kuten kuvasta 2 näkee, Npm:n riippuvuusjärjestelmän sisäkkäisyys tekee järjestelmästä puumaisen. Tämän hyvä puoli on se, että riippuvuudet eivät aiheuta konflikteja keskenään. Haittapuolena on se, että sisäkkäisissä riippuvuuksissa voi tarpeettomasti olla useampi versio jostakin kirjastosta. Bowerin haittana taas on se, että konflikteja esiintyy ja käyttäjän täytyy ratkaista ne itse. [10.]

4.2 Sovellus

4.2.1 HTML5 Canvas

HTML5 Canvas on HTML-standardin mukainen piirtoalustaelementti [11]. Sille voi piirtää viivoja, suorakulmioita ja monimutkaisempia kuvioita kuten bezier-kurveja. Myös tekstin ja kuvien piirtäminen sekä koko kontekstin muunnokset ovat mahdollisia. Canvas-elementti on laajasti tuettu ja mikä tärkeintä, se toimii ja näyttää samalta selaimesta riippumatta.

Canvasille on keksitty monia eri käyttötarkoituksia mm. dynaamisten grafiikkojen piirtäminen, selainpohjaiset pelit, animaatiot ja jopa sivuston kävijöiden ns. sormenjälkitunnistus [12; 13].

Työssä päädyttiin käyttämään Canvasia siitä syystä, että se näyttää samalta eri selaimissa ja päätelaitteissa. Siinä ei myöskään ole samanlaisia tietoturvaongelmia kuten vaikkapa Adobe Flashissä, koska se on HTML-standardin mukainen elementti. Canvasin ansiosta Teksti-TV-sovelluksen käyttäjä ei tarvitse muuta kuin selaimen, jossa Javascript on sallittu.

Canvasin käyttäminen on hyvin yksinkertaista. Canvas-elementti upotetaan HTML-sivulle ja Javascriptillä haetaan tästä elementistä piirtokonteksti getContext()-metodilla.

4.2.2 Javascript

Javascriptiä varten on olemassa erittäin paljon hyödyllisiä avoimen lähdekoodin kirjastoja, joiden hyödyntäminen onnistuu nykyään helposti paketinhallintajärjestelmien ansiosta.

Taulukko 1. Työssä käytetyt kirjastot.

Kirjasto	**Versio**
Backbone	1.2.3
Bootstrap	3.3.4
Del	2.0.2
Gulp	3.9.0
Gulp-uglify	1.4.1
JQuery	2.1.4
RequireJS	2.1.20
Underscore	1.8.3

Backbone-kirjasto valittiin käytettäväksi siitä syystä, että se helpottaa sovelluksen yhdistämistä REST-rajapintaan ja auttaa ketä tahansa Backbonen mallia ja näkymää tuntevaa ymmärtämään sovelluksen rakennetta. Backbone sopi työhön parhaiten siksi, että se on

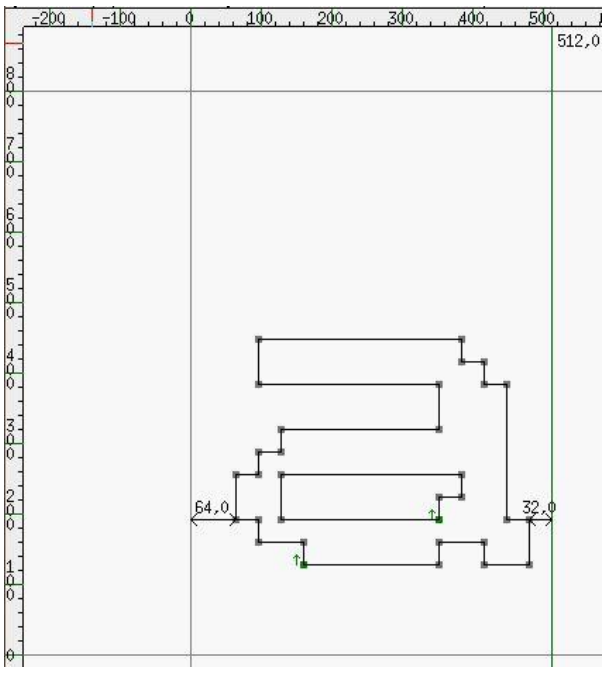
filosofialtaan minimalistinen. [14] Sovellus ei myöskään tarvinnut tuekseen monimutkaista ohjelmistokehystä. Underscore-kirjasto osaltaan on Backbonen riippuvuus, joka sisältää hyödyllisiä funktioita olioiden, funktioiden ja taulukoiden muokkaamiseen.

Bootstrap-kirjasto on erittäin suosittu avoimen lähdekoodin responsiivisten käyttöliittymien kehittämiseen käytetty ohjelmistokehys. Se valittiin työhön responsiivisten ominaisuuksiensa ja valmiiden komponenttiensa ansiosta. Jquery on sen riippuvuus, mutta sitä käytettiin laajasti myös muualla työssä.

Muut taulukossa 1 listatut kirjastot liittyvät projektin kokoamiseen, ja ne esitellään luvussa 4.3.

4.2.3 Fontti

Fontti luotiin avoimen lähdekoodin ohjelmalla nimeltä FontForge. FontForgella merkistö luodaan hahmottelemalla jokaisen yksittäisen merkin ääriviivat (katso kuva 3). Sovelluksen fontti perustuu Unicode-merkistöön, koska editori sisältää paljon erikoismerkkejä, joita ei ole olemassa missään olemassaolevissa merkistöissä. Unicode kuitenkin sisältää vapaana olevia rekistereitä, joten erikoismerkit oli järkevää sijoittaa sellaiseen.



Kuva 3. FontForge-ohjelman muokkausnäkymä.

Fontin tiedostomuodoksi olisi haluttu SVG (Scalable Vector Graphics), mutta valitettavasti selainvalmistajat ovat luopumassa SVG-fonttien tukemisesta. Tästä syystä fontin tiedostomuodoksi valittiin TrueType, joka on edelleenkin eräs tuetuimmista fonttien tiedostomuodoista eri selaimissa. Vektorifontti olisi saattanut näyttää paremmalta skaalattuna kuin TrueType-fontti.

4.3 Kokoaminen

RequireJS-kirjasto on tarkoitettu Javascript-tiedostojen ja -moduulien tuomiseen. Kirjastoa käyttävälle HTML-sivulle lisätään vain viittaus RequireJS-konfiguraatiotiedostoon. Konfiguraatiotiedostossa määritellään muut Javascript-tiedostot (ideaalitilanteessa yksi tiedosto sisältää yhden moduulin), jotka RequireJS liittää sivulle. Tämä tuo sovellukseen yksinkertaisen rakenteen ja mahdollistaa helpon kasaamisen. RequireJS-moduulit perustuvat Javascriptin Moduuli-suunnittelumalliin [15].

RequireJS sisältää oman minifiointiin ja tiedostojen yhteenliittämiseen tarkoitetun komponentin. Sitä käyttämällä projekti voidaan kasata yhdeksi isoksi minifioiduksi Javascript-tiedostoksi. Minifiointi pienentää projektin koon murto-osaksi siitä, mitä se on kehitysvaiheessa.

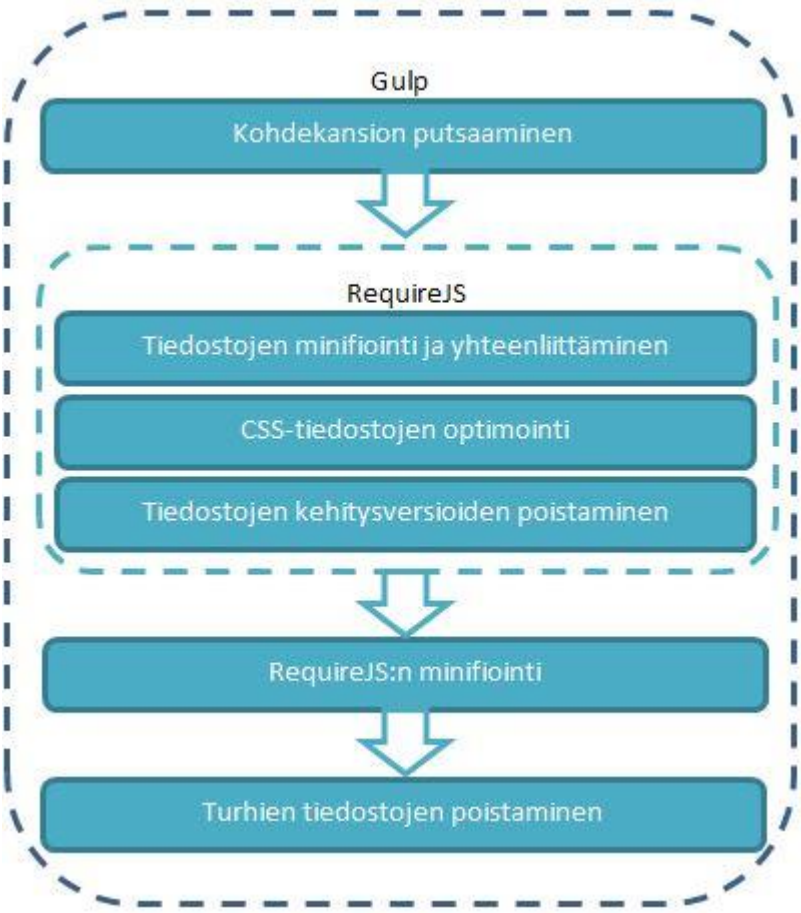
Gulp on Javascript-projektien kokoamiseen tarkoitettu kirjasto. Se sisältää myös muita ominaisuuksia, mutta tässä työssä sitä käytettiin ainoastaan projektin kokoamiseen. RequireJS ei osaa minifioida itseään, joten Gulp on pätevä työkalu tätä varten. Taulukossa 1 mainitut Del- ja Gulp-uglify-kirjastot ovat myös osa projektin kokoamisjärjestelmää.

Gulp toimii Node.js-ajoympäristössä. Gulpin kokoamistiedosto koostuu tehtävistä (engl. *task*). Tehtävän parametrit ovat nimi, riippuvaisuudet ja käsittelijäfunktio. Oletusarvoisesti kaikki tehtävät suoritetaan rinnakkain [16].

```
11  gulp.task('r', ['clean'], function(cb) {
12    exec('node r.js -o js/ttveditor.build.js',
13      function (err, stdout, stderr) {
14        console.log(stdout);
15        console.log(stderr);
16        cb(err);
17    });
18  });
```

Kuva 4. Esimerkki Gulpin tehtävästä.

Jos tehtäviä haluaa ajaa sarjassa, pitää käyttää Javascriptissä yleisesti käytettyä *call-back*-menetelmää. Kuten kuvasta 4 voi nähdä, tehtävän käsittelijäfunktio ottaa vastaan *callback*-funktion, jota kutsutaan, kun tehtävä on suoritettu. Kuvasta 4 voi myös nähdä, kuinka RequireJS-kirjaston oma minifiointiin käytetty komponentti käynnistetään. Tämä on mahdollista, koska Node.js-ajoympäristöä voi ajaa lapsiprosessina. Kokoamistiedosto ajetaan komentorivillä komennolla *gulp*.



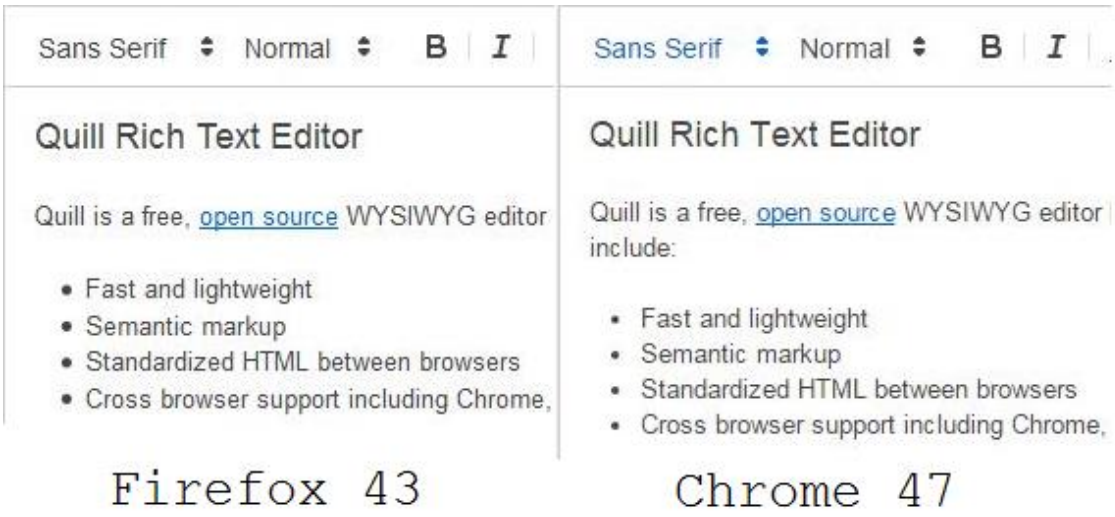
Kuva 5. Kokoamisjärjestelmän vaiheet.

Kuvassa 5 on esitetty kokoamisjärjestelmän eri vaiheet ja missä järjestyksessä ne suoritetaan. Kokoamisen ansiosta projektin Javascript-tiedostojen lopullinen koko on alle 40 % verrattuna kehitysversioon.

Sovelluksen minifiointi on järkevää tehdä, koska sivunlatausten määrän kasvaessa se pienentää oleellisesti palvelimen käyttämää kaistaa (eli säästää rahaa) ja koska se lyhentää käyttäjän selaimen Javascriptin parsimiseen kuluttamaa aikaa. Javascript-tiedostojen yhdistäminen myös takaa sen, että selaimen täytyy ladata vain yksi Javascript-tiedosto, joka sekin osaltaan nopeuttaa latausprosessia. Sovellus siis latautuu nopeammin pienemmillä resursseilla kokoamisen ansiosta.

5 Toteutus

Tässä luvussa käydään läpi sovelluksen toteuttamisessa esiin tulleita asioita ja ongelmia. Toteuttaminen aloitettiin käytettävien teknologioiden valitsemisella, jotka on esitelty luvussa 4. Hyvin aikaisessa vaiheessa ilmeni, että editorin toteuttaminen jollakin olemassaolevalla WYSIWYG-editorilla olisi hyvin nopeaa, mutta lopputulos ei näyttäisi samalta eri selaimilla.



Kuva 6. Quill Firefoxissa ja Chromessa.

Kuvassa 6 on vertailtu miltä WYSIWYG-editori nimeltä Quill näyttää Firefoxissa ja Chromessa. Ongelma on samanlainen kaikissa WYSIWYG-editoreissa, koska eri selaimet

laskevat elementtien leveydet eri tavalla. Ongelma myös pysyy samana, oli editorin pohjana textarea-komponentti tai contenteditable-määre. Kuvasta 6 voi nähdä, kuinka sana include ei mahdu Chromessa samalle riville kuin Firefoxissa. Kyseessä ei ole se, että fontti eroaisi selainten kesken, sillä ongelma on sama myös monospace-fonttien kanssa (monospace-fontissa kaikki merkit ovat samanlevyisiä).

Tässä vaiheessa on hyvä todeta, että HTML5 Canvasin standardi varoittaa toteuttamasta tekstieditoria Canvasille [17]. Teksti-TV-editorin tavoitteena kuitenkin oli näyttää samalta eri selaimissa, joten jäljelle ei jäänyt muuta vaihtoehtoa kuin HTML5 Canvas. Kyse ei myöskään ole yleisluontoisesta kaikenlaisen tekstin muokkaamiseen soveltuvasta editorista. Canvasille olemassaolevat tekstin muokkaamiseen tarkoitetut kirjastot Fabric.js (sisältää komponentin *fabric.Text*) ja Carota osoittautuivat liian hitaiksi, joten tästäkin syystä oli parasta aloittaa sovelluksen kehittäminen puhtaalta pöydältä.

Teksti-TV-sivu sisältää 23 riviä, joilla jokaisella on 40 merkkiä. Editori sisältääkin siis 920 solua, jotka sisältävät tarpeelliset tiedot kuten merkin, merkin värin, merkin koon ja solun taustavärin. Tekstikursori osoittaa nykyisen paikan, johon merkki voidaan asettaa. Tekstikursoria voi liikuttaa nuolinäppäimillä, hiiren painalluksella tai kosketusnäyttöä painamalla.

5.1 Merkkitoiminnot

Teksti-TV-editorin käyttäjä voi syöttää, poistaa, valita, leikata, kopioida ja liittää merkkejä editoriin. Käyttäjän toiminnot havaitaan tapahtumankuuntelijoilla. Kun käyttäjä syöttää merkin, alkaa monimutkainen prosessi. Teksti-TV:n tausta- ja merkkivärit sekä muut ominaisuudet osoitetaan kontrollimerkeillä. Kun käyttäjä syöttää tällaisen kontrollimerkin, pitää koko rivi tarkistaa ja päivittää, sillä kontrollimerkit vaikuttavat koko riviin. Sama pitää myös tehdä, jos käyttäjä poistaa riviltä kontrollimerkin.

Merkkejä voi valita Shift + nuolinäppäimillä, hiirellä maalaamalla tai kosketusnäytöllä sormea liikuttamalla. Valittu alue havainnollistetaan vaaleansinisellä huomiovärillä (katso kuva 7). Vanhassa editorissa valinta toimi pelkästään oikealle alas. Uudessa editorissa tätä ominaisuutta kehitettiin siten, että valinta toimii nyt kaikkiin suuntiin. Valintaneliö piirretään vain silloin, kun valintaneliö on muuttunut edelliseen tapahtumaan verrattuna. Valintaneliö muuttuu vain silloin, kun osoitin liikkuu solusta toiseen.

Kuva 7. Valittu alue.

Valitun alueen sisällön voi poistaa, leikata tai kopioida. Leikkaaminen ja kopiointi siirtävät valittujen solujen sisällön editorin leikepöydälle, jonka sisällön voi liittää joko samalle alasivulle tai jollekin toiselle.

5.2 Vieritys

Vierityksellä tarkoitetaan sitä, että teksti-tv:n sivulla rivillä jo olevat merkit siirtyvät uusia merkkejä syötettäessä eteenpäin tai merkkejä poistettaessa taaksepäin. Editoriin toteutettiin kaksi erilaista vieritystä: ”vieritys vain riville” ja ”vieritys sivun loppuun asti” (kuva 8).

”Vieritettäessä vain riville” merkit katoavat mennessään oikean reunan yli. ”Vieritettäessä sivun loppuun asti” merkit siirtyvät oikean reunan yli mennessään seuraavalle riville poikkeuksena tietenkin viimeinen rivi.

Kuva 8. Käyttäjää informoidaan tekstillä ja värillä siitä, kumpi vieritys on käytössä.

Yleisesti ottaen vieritys toimii seuraavasti: kun käyttäjä syöttää merkin, etsitään viimeinen vieritettävä merkki. Sillä, onko merkki samalla rivillä vai jollakin alemmalla rivillä, ei ole väliä, sillä periaate on aina sama. Tästä viimeisestä merkistä lähtien siirrytään vasemmalle päin siten, että vieritettävä merkki siirretään yhden solun verran oikealle. Toisin sanoen vieritettäessä merkkejä eteenpäin siirrytään lopusta alkuun ja vieritettäessä taaksepäin, siirrytään luonnollisesti päinvastoin alusta loppuun.

5.3 Rivitys

Eräs työn haastavimpia osa-alueita oli rivityksen toteuttaminen. Rivityksellä tarkoitetaan teksti-tv:n sivulla olevien sanojen siirtämistä dynaamisesti sopiviin paikkoihin suhteessa sivun leveyteen. Tämä tulee eteen, kun käyttäjä syöttää merkkejä, jotka puskevat sanan oikeasta reunasta yli, mutta sana halutaan pitää koossa (toisin sanoen, sanaa ei haluta vierittää).

Toisin kuin vierityksessä, jossa sana puskettaisiin kirjain kerrallaan seuraavalle riville, rivityksessä koko sana siirretään kerrallaan seuraavalle riville. Tämä taas vaikuttaa todennäköisesti myös muihin seuraavalla rivillä oleviin sanoihin ja sitä seuraaviin riveihin jne.

```
SpaceLeft := LineWidth
for each Word in Text
    if (Width(Word) + SpaceWidth) > SpaceLeft
        insert line break before Word in Text
        SpaceLeft := LineWidth - Width(Word)
    else
        SpaceLeft := SpaceLeft - (Width(Word) + SpaceWidth)
```

Kuva 9. Rivityksessä käytetty algoritmi [18].

Ongelman ratkaisemiseksi hyödynnettiin olemassaolevaa algoritmia. Kuvasta 9 voi nähdä käytetyn algoritmin pseudokoodin. Kyseinen algoritmi käyttää rivittämiseen mahdollisimman vähän rivejä. Algoritmi on hyvin yksinkertainen, mutta on silti tarpeeksi hyvä tähän käyttötarkoitukseen. Se syöttää sanoja riville, kunnes rivillä ei ole enää tilaa, jolloin se siirtyy seuraavalle riville. Sanojen väliin tulee välimerkki.

Rivityksessä on kyse optimointiongelmasta. Miten saadaan aikaan algoritmi, joka on nopea ja rivittää vain ja ainoastaan ne sanat, jotka on pakko rivittää? Vaikein osa algoritmin tekemistä oli varsinaisten sanojen syöttämisen sijasta kaikki muu, joka algoritmin toimintaan kuului.

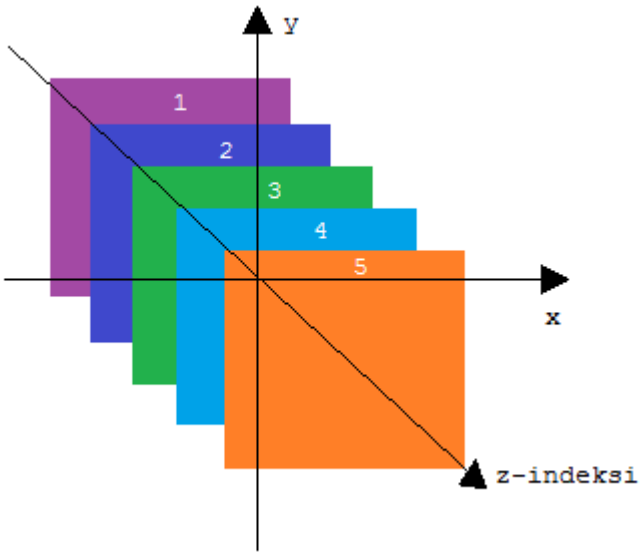
Rivityksessä kestää sitä kauemmin, mitä enemmän on rivejä ja rivitettäviä sanoja. Jotta algoritmista saadaan nopea, pitää selvittää myös sana (ts. solu), johon pitää lopettaa,

koska ei ole mitään järkeä rivittää uudestaan sanoja, jotka vain menisivät takaisin samoille paikoille. Rivitettävä alue pitää myös putsata, jotta rivitettävät sanat eivät sotkeennu omiin kopioihinsa. Tämä kaikki oli helpommin sanottu kuin tehty ja tämän ominaisuuden toteuttamiseen kului todella paljon aikaa.

5.4 Apupiirtoalusta

Editori sisältää myös toisen HTML5 Canvas -piirtoalustan, jolle piirretään tekstikursori ja apuverkko. Nämä on erotettu toiseen piirtoalustaan, koska ne täytyy piirtää vain silloin, kun ne muuttuvat. Tämä vähentää oleellisesti piirrettävien pikseleiden määrää ja nopeuttaa sovelluksen suorittamista.

Apuverkko koostuu 920 pisteestä, jotka piirretään jokaisen solun vasempaan yläkulmaan. Jos solun sisältämää pistettä ei piirrettäisi erilliselle piirtoalustalle, pitäisi piste piirtää joka kerta, kun solun sisältö muuttuisi. Tekstikursori ei vie paljon resursseja, mutta myös se kannattaa piirtää muulle kuin varsinaiselle piirtoalustalle, koska silloin se täytyy piirtää vain, kun sen paikkaa muutetaan. Apupiirtoalusta on läpinäkyvä ja z-indeksin perusteella – Z-indeksi tarkoittaa vektoria, joka nousee sivun perältä kohti katsojaa (kuva 10) – varsinaisen piirtoalustan päällä, joten alla olevan varsinaisen piirtoalustan muutokset eivät vaikuta apupiirtoalustaan.



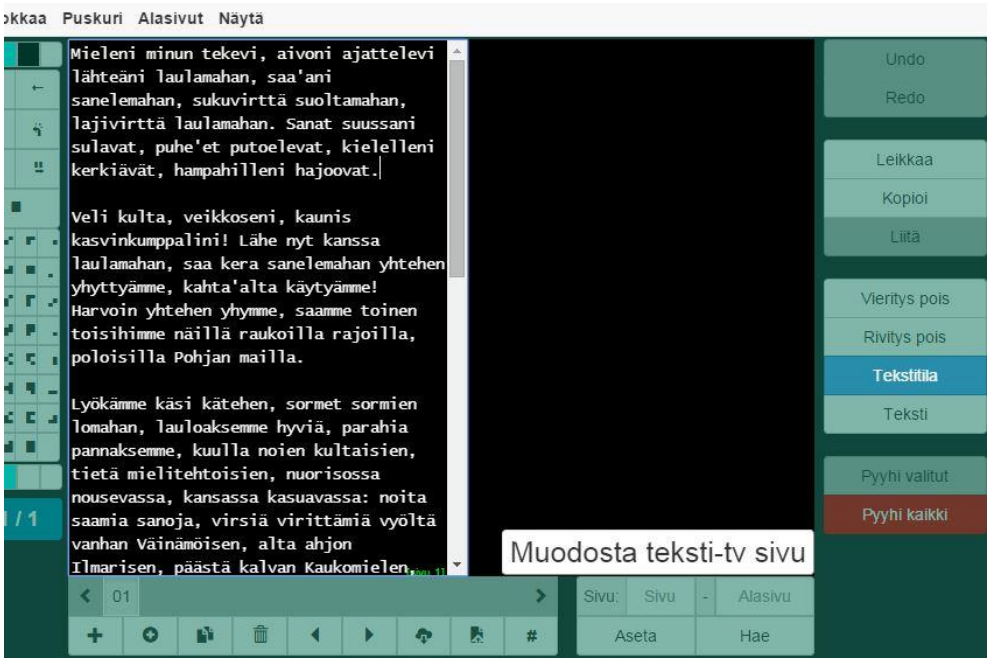
Kuva 10. HTML-elementtien z-indeksi. Suurin z-indeksi tulee pinon päälimmäiseksi.

5.5 Tekstitila

Aikaa kenties eniten säästävin toiminto oli tekstitila. Se koostuu tekstieditorin näyttämisestä ja muiden toimintojen lukitsemisesta. Toisin kuin tavallinen editori, se ei näytä lopputulosta reaaliaikaisesti, vaan mahdollistaa tekstin muokkaamisen, kopioimisen ja poistamisen useille alasivuille yhtä aikaa.

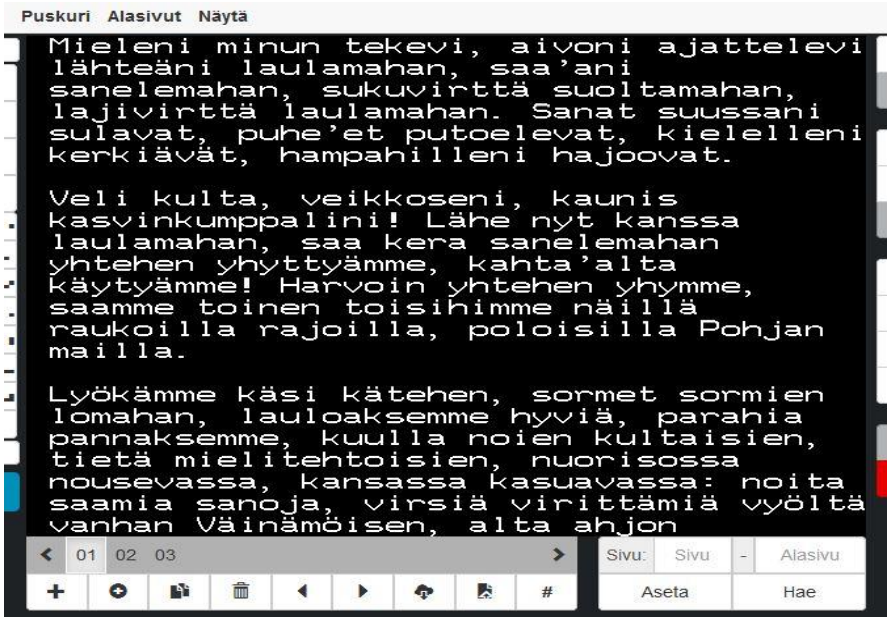
Kun käyttäjä painaa tekstitilan painiketta, joko valikosta tai oikealle sijoitetusta painikemenusta, näytetään tekstieditori tavallisen editorin päällä, ja kaikki muu paitsi valikko ja oikean painikemenun tekstitilan painike lukitaan.

Teknisesti tämä toteutettiin näyttämällä tätä tarkoitusta varten luotu läpinäkyvä ns. taustakangas (engl. *backdrop*) ja määrittämällä sivun z-indeksit siten, että ne joko tulevat taustakankaan päälle tai taakse. Taustakankaan idea lainattiin Bootstrapin modaalikomponentista. Bootstrapin valmista modaalia ei käytetty, koska vanhan käyttöliittymän mukaisen ratkaisun tekeminen oli helppoa.



Kuva 11. Tekstitila.

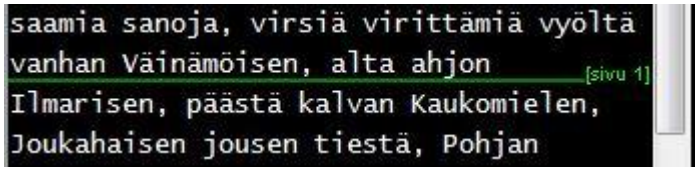
Kuvassa 11 näkyy tekstitila. Lukitut komponentit näkyvät turkoosin taustakankaan takana eikä niitä voi painaa. Tekstitilasta pääsee pois painamalla tekstitilapainiketta joko yläpalkin Näytä-valikosta tai oikeasta painikevalikosta (oikean painikevalikon tekstitilapainike on havainnollistettu sinisellä huomiovärillä.) Tekstieditori on textarea-elementti. Tekstieditori sisältää normaalit tekstin muokkaustoiminnot.



Kuva 12. Tekstitilan generoima Teksti-TV-sivu.

Käyttäjän painaessa Teksti-TV-sivun luontipainiketta sivut generoidaan automaattisesti. Kuvan 12 tapauksessa generoituja alasivuja on kolme kappaletta. Kuten kuvasta 12 voi nähdä, on sivua generoitaessa käytetty aikaisemmin kuvattua rivitysalgoritmia.

Tekstitilaa tehtäessä havaittiin lukuisia rajoitteita ja ongelmia. Kuten vanhakin editori, myös nykyinen tekstitilan tekstieditori sisältää apuviivan, joka näyttää, missä kohtaa sivu vaihtuu (kuva 13).



Kuva 13. Sivun apuviiva.

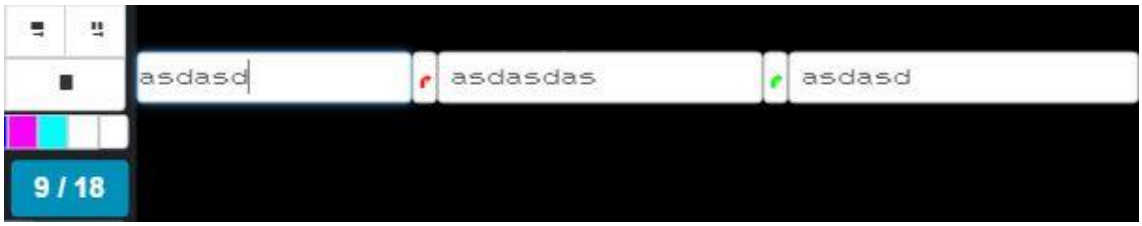
Tämän saaminen täysin toimivaksi vaatisi selainkohtaisia optimointeja, koska selaimet laskevat HTML-elementtien leveydet eri tavalla. Teksti ei täten rivity samalla tavalla eri selaimissa, joten apuviiva ilmestyy väärään paikkaan suhteessa tekstiin. Ainoa tapa korjata tämä tällä hetkellä olisi korvata textarea-elementti omalla Canvas-toteuksellaan, joka veisi paljon aikaa verrattuna siitä saatavaan hyötyyn.

5.6 Kosketustoiminnot

Editoriin toteutettiin myös toiminnallisuus tekstin syöttämiseen kosketuslaitteella. Kun käyttäjä koskettaa näyttöä pitkään joltakin riviltä, generoidaan riviltä olevasta tekstistä HTML input -elementti, jota käyttäjä pääsee muokkaamaan. Jokaista väriä varten generoidaan oma kenttä. Syynä on, että yhdelle input-elementille voi määritellä vain yhden värin kerrallaan.

Kuva 14. Teksti ennen kuin kosketustoimintoa on käytetty.

Kuvassa 14 on havainnollistettu kosketustoiminnallisuus. Kosketustoiminnallisuus ei aiheuta minkäänlaisia muutoksia editoriin, vaan teksti näyttää aivan samalta kuin normaalistikin.



Kuva 15. Kosketusnäytöillä toimivat syöttökentät.

Kuvassa 15 näkee, miltä tekstin syöttöä varten generoidut kentät näyttävät. Kosketuslaitteen graafisen näppäimistön saa esille koskettamalla jotakin kenttää. Tekstin muokkaaminen toimii laitteesta riippumatta normaalisti, koska kyseessä on pelkkä HTML input -elementti. Työn aikana ei löydetty muita tapoja tuoda kosketuslaitteen graafinen näppäimistö näkyviin.

5.7 Alasivut

Jokainen kantasivu voi sisältää useita alasivuja. Standardi mahdollistaa jopa 3200 alasivua, mutta editorissa määrä on rajoitettu 99:ään [2]. Alasivut eivät eroa kantasivuista teknisesti mitenkään, eli ne käytännössä sisältävät vain oman piirtoalustansa. Editorin kannalta jokainen sivu on prototyyppisesti sama, vain sisältö eroaa.

Kuva 16. Alasivujen työkalupalkki.

Käyttäjä voi luoda uusia alasivuja ennen valittua alasivua tai loppuun, sekä kopioida, siirtää ja poistaa niitä. Alasivut voi myös numeroida. Siirtäminen onnistuu vasemmalle ja oikealle, ja myös käyttäjän valitsemaan kohtaan. Alasivujen työkalupalkkiin mahtuu samanaikaisesti kahdentoista alasivun valintapainikkeet (katso kuva 16).

Editorin näkymä pitää kirjaa alasivujen lukumäärästä. Valittuna oleva alasivu ilmoitetaan huomiovärillä, joka saadaan aikaiseksi liittämällä HTML-elementtiin tätä varten määritelty luokkamääre. Luokkamäärettä käytetään myös Javascript-koodissa selvittämään, mikä alasivu kulloinkin on valittuna. Tätä tarvitaan esimerkiksi, kun käyttäjä haluaa lisätä alasivun ennen valittua alasivua. On parasta, että tieto kulloinkin valittuna olevasta alasivusta on vain yhdessä paikassa. Tieto voikin siis yhtä hyvin olla valitun HTML-elementin CSS-luokkamääre, kunhan Javascript-koodissa pidetään huoli, että tämä luokka on käytössä vain yhdellä elementillä kerrallaan.

5.8 Muokkaushistoria

Muokkaushistorian tekeminen oli kenties vaikein osa-alue työssä. Nopean ja luotettavan muokkaushistorian tekemiseen on monia eri tapoja. Suurimmat ongelmat muokkaushistoriassa tulevat siitä, että kulloinkin historiaan syötettävän merkkijonon kokoa ei voida tietää etukäteen, mikä voi tehdä historiasta hitaan. [19.] Koska kyseessä oli editori, jonka tilan koko on jokaisella hetkellä sama (960 solua), voitiin muokkaushistoriasta tehdä yksinkertainen.

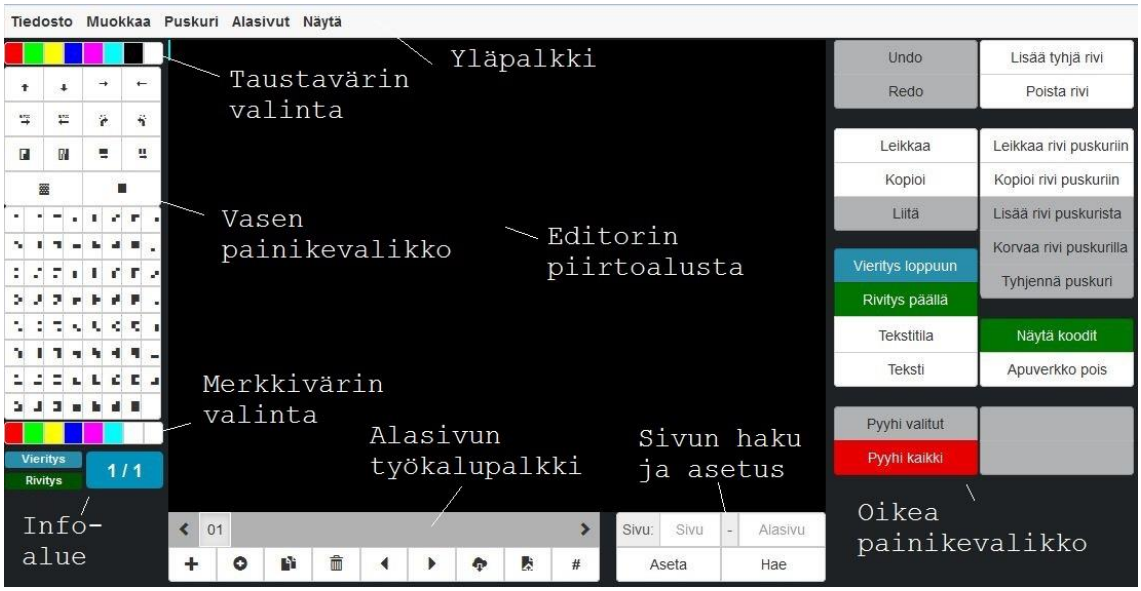
Käyttäjän toimet laukaisevat historiatapahtumia. Esimerkiksi kun käyttäjä syöttää merkkejä ja siirtyy seuraavalle riville, puskurissa olevista muutoksista tehdään palautuspiste muokkaushistoriataulukkoon. Käyttäjä voi liikkua muokkaushistoriassa perumalla ja ”tekemällä uudelleen” muutoksia. Näkymässä nämä toiminnot ovat Undo- ja Redo-painikkeet ja pikanäppäimissä Ctrl + Z ja Ctrl + Shift + Z.

Alkuperäisessä editorissa vain yhden muutoksen pystyi perumaan kerrallaan. Tästä syystä katsottiin, että uuden toteutuksen ei tarvitse olla merkittävästi parempi. Historian olisi voinut varmasti toteuttaa paremminkin, mutta yllättävää kyllä riittävällä määrällä optimointeja tämäkin versio toimii erinomaisesti lähes täydellä 60 fps:n ruudunpäivityksellä. Hypoteettisesti muokkaushistorian täytyy ruveta hidastumaan sitä mukaa mitä enemmän palautuspisteitä historiaan syötetään. Testauksen aikana ei kuitenkaan löydetty tällaista hidastumispistettä, joka johtuu luultavasti siitä, että nykyiset tietokoneet ovat niin tehokkaita, ja että tavallisessa käytössä valtavan palautuspistemäärän syntyminen on epätodennäköistä, koska Teksti-TV-sivut ovat niin yksinkertaisia.

Muutoksien ”tekeminen uudestaan” on yksinkertaista, koska editorin nykyinen tila on jo oikea ja seuraavan palautuspisteen muutokset voidaan suoraan piirtää nykyisen tilan päälle. Muutosten peruminen on hankalampaa, koska edellinen palautuspiste sisältää vain muutokset sitä edelliseen palautuspisteeseen ja niin edelleen muokkaushistorian alkuun asti. Tämän toteuttaminen paremmin jäi tekemättä, koska yllättäin muokkaushistoria toimi tälläkin tavalla erinomaisesti.

5.9 Käyttöliittymä

Editorin käyttoliittymän pohjaksi valittiin Bootstrap-kirjasto. Se on erittäin suosittu Javascript-ohjelmistokehys responsiivisten käyttöliittymien suunnittelemiseen. Bootstrap-kirjastolla kehitys aloitetaan ikään kuin mobiililaitteista käsin. Sen CSS-mediahaut on suunniteltu siten, että oletusarvoina ovat pieniruutuiset laitteet ja isompiruutuiset laitteet ylikirjoittavat CSS-määreitä. Bootstrap-kirjaston ja käyttöliittymän responsiivisuutta käsitellään tarkemmin seuraavassa luvussa.



Kuva 17. Käyttöliittymä.

Käyttöliitymä (kuva 17) koostuu yhdeksästä komponentista. Yläpalkki on Bootstrap-kirjaston navbar-komponentti, joka muistuttaa vanhaa käyttöliittymää. Se sisältää kaikki tärkeimmät toiminnot. Vasen painikevalikko sisältää taustavärien ja merkkivärien valinnan, sekä kontrollimerkki- ja grafiikkamerkkipainikkeet. Editorin piirtoalusta on musta alue keskellä, jolle Teksti-TV-sivu luodaan.

Oikea painikevalikko sisältää hyödyllisimmät toiminnot, kuten leikkaa, kopioi, liitä, vierityksen ja rivityksen päälle / pois painikkeet, sekä kontrollimerkkien ja apuverkon näyttämisen painikkeet. Oikea painikevalikko on olemassa, jotta käyttäjä voisi yhdellä painalluksella käyttää yleisesti käytettyjä toimintoja. Toiminnot löytyvät myös yläpalkin valikoista, mutta niiden käyttäminen vaatii kaksi painallusta. Bootstrapin valikko pitää oletuksena avata ja sulkea painalluksella. Oikea painikevalikko piilotetaan, mikäli ikkuna tai ruutu on liian kapea sen näyttämiseen.

Infoalue vasemmassa alakulmassa näyttää, ovatko vieritys, rivitys tai puskuri käytössä. Vierityksen tyypistä informoidaan käyttäjää väreillä. Vihreä tarkoittaa, että käytössä on ”vieritys riville” ja vaaleansininen, että käytössä on ”vieritys loppuun”. Infoalue näyttää aina myös sen solun, jossa tekstikursori on, formaatilla: rivinumero / sarakenumero.

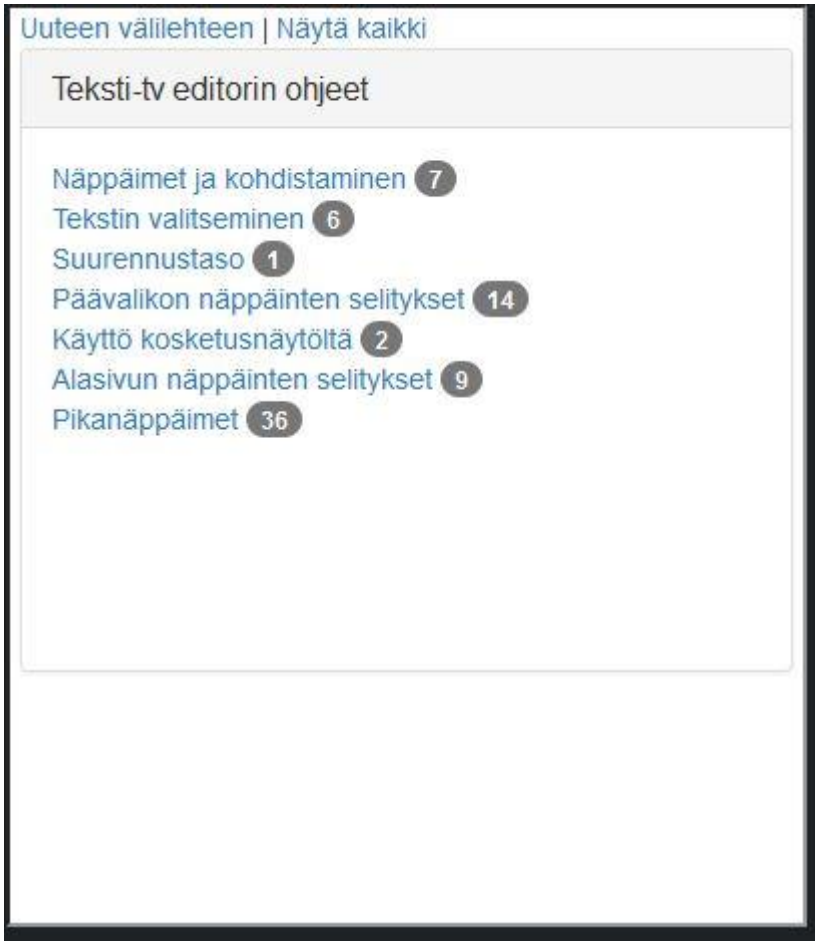
Alasivun työkalupalkki sisältää alasivujen tärkeimmät toiminnot. Ne on eritelty tarkemmin luvussa 5.7 (katso myös kuva 16). Sivun haku ja asetus -komponentti sisältää kentät, joihin syötetään sivunumero ja mahdollisesti haluttu alasivun numero. Hae-painike hakee halutun sivun ja Aseta-painike asettaa nykyisen editorin sivun.

Kuva 18. Muokkaa-valikko.

Yläpalkin valikoita muokattiin sopimaan paremmin sovellusta varten. Painikkeista tehtiin erikokoisia, jotta niitä voitiin laittaa riveille eri määriä, kuten kuvasta 18 voi nähdä. Tämä

tehtiin helpottamaan kosketusnäytön käyttäjiä. Testattaessa valikoita havaittiin, että liian pienet painikkeet johtivat helposti väärän painikkeen painamiseen. Valikkojen painikkeiden suuruudesta ei ole haittaa hiiren käyttäjille, joten ratkaisu oli toimiva tästäkin syystä.

Editorin käyttöliittymä sisältää monia huomiovärejä. Harmaa painike tarkoittaa, että painike on poissa käytöstä. Esimerkiksi Undo- ja Redo-painikkeet ovat aina sovelluksen ensikäynnistyksen tai sivun päivittämisen jälkeen pois käytöstä, koska muokkaushistoria on tyhjä. Painikkeet myös informoivat käyttäjää siitä, onko käyttäjä saavuttanut ensimmäisen tai viimeisen muokkaushistorian pisteen. Samalla tavalla Liitä-painike ei toimi, jos editorin leikepöytä on tyhjä. Vihreä tai vaaleansininen huomioväri tarkoittaa, että jokin ominaisuus on kytketty päälle. Punainen huomioväri tarkoittaa, että toiminto on peruuttamaton ja sen käyttämistä kannattaa miettiä tarkkaan. Käyttäjältä myös kysytään erikseen, haluaako hän suorittaa tämän toiminnon, kun punaista painiketta on painettu.



Kuva 19. Editorin käyttöohje.

Editori sisältää myös käyttöhjeen (kuva 19), jonka saa avattua Näytä-valikosta. Se avautuu alimmaiseksi elementiksi sivulla, jossa sitä voi selata. Sen voi myös avata uuteen välilehteen. Käyttöohje sisältää tärkeimmät tiedot, joita tarvitaan editorin käyttämiseksi. Se sisältää myös listauksen editorin pikanäppäimistä. Käyttöohje on hyvä olla olemassa, jotta käyttäjä saa helposti selville, kuinka ohjelmaa käytetään ja minkälaisia toimintoja se sisältää.

Käyttöliittymän suunnitteluun ja toteutukseen meni eniten aikaa koko työssä. Käyttöliittymä muuttui kehityksen aikana muutamalla ratkaisevalla tavalla. Kun responsiivisia ominaisuuksia ryhdyttiin suunnittelemaan, kävi selväksi, että sen aikainen käyttöliittymä ei olisi toiminut niiden kanssa kovin hyvin. Käyttöliittymästä muokattiin enemmän vanhan käyttöliittymän mukainen, mutta parannuksia ei kuitenkaan unohdettu. Erisuuruisten ruutujen huomioiminen käyttöliittymän suunnittelussa tarkoitti sitä, että joitakin kompromisseja oli tehtävä.

5.10 Responsiivisuus

Responsiivisuudella tarkoitetaan HTML-sivun kykyä mukautua päätelaitteeseen. Responsiivisuuden ansiosta HTML-sivusta ei tarvitse luoda eri versiota mobiililaitteita varten, vaan sivusta luodaan yksi versio, joka muuttuu dynaamisesti näytön leveyden mukaan. Sivun komponentit esimerkiksi suurenevat tai pienenevät sen mukaan, miten paljon näytöllä on tilaa.

Sovelluksen responsiivisuus toteutettiin Bootstrap-kirjastoa hyödyntäen. Bootstrap-kirjasto perustuu ruudukkoon, jossa on kaksitoista saraketta. Sarakkeiden leveys vaihtelee näytön leveyden mukaan. Tämä mahdollistaa sen, että käyttöliittymästä voidaan tehdä useita eri versioita, jotka tulevat dynaamisesti käyttöön näytön leveyden muuttuessa. Tyypillisesti tällaista tapahtuu esimerkiksi, kun käyttäjä kääntää tabletin tai älypuhelimen vaakatasosta pystyyn tai kun käyttäjä muuttaa selainikkunan leveyttä.

Miksi Bootstrap huomioi ainoastaan leveyden? Siitä syystä, että nettisivuja tyypillisesti kelataan ylhäältä alas eikä vasemmalta oikealle. Nettisivun pituuden mittaaminen on myös tehty todella hankalaksi eikä ole mitään takeita, että se on sama kaikissa selaimissa.

Taulukko 2. Bootstrapin ruudukkojärjestelmä [20].

	Erittäin pienet laitteet	**Pienet laitteet**	**Keskikokoiset laitteet**	**Suuret laitteet**
Näytön leveys	< 768 px	≥ 768 px	≥ 992 px	≥ 1200 px
Säiliön leveys	automaattinen	750 px	970 px	1170 px
Sarakkeen leveys	automaattinen	62 px	81 px	97 px

Taulukossa 2 on listattu ruudukkojärjestelmän tärkeimmät mitat. Kaikki elementit, joihin ruudukon halutaan vaikuttavan, laitetaan yhden pääsäiliön sisälle. Säiliön molemmille puolille jätetään yhteensä 30 pikseliä tyhjää tilaa. Tämän asetuksen, kuten kaiken muunkin, voi muokata mieleisekseen.

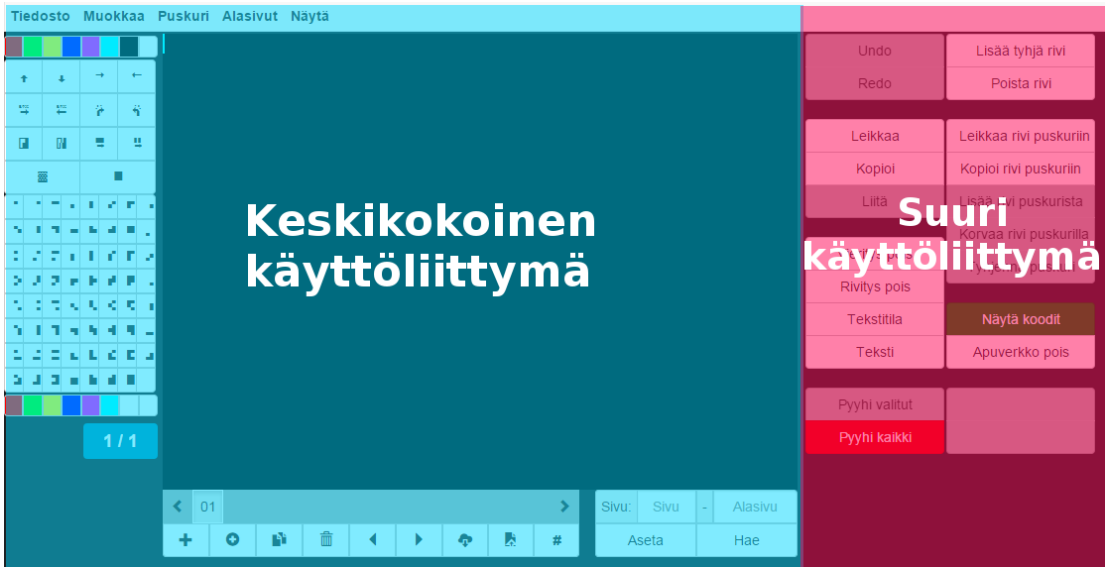
Bootstrap käyttää CSS:n mediahakuja muuttamaan käyttöliittymän ulkoasun tietylle näytön leveydelle sopivaksi. Esimerkiksi navbar-komponentti, jota käytetään valikkojen tekemiseen, menee erittäin pieniruutuisella laitteella kasaan yhden painikkeen taakse. Bootstrap on suunniteltu mobiililaitteet edellä, eli erittäin pienten laitteiden tyyliohjeet ovat oletuksena ja leveyden kasvaessa mediahaut ylikirjoittavat niitä.

Työssä käytettiin laajasti painikkeita, painikeryhmiä ja navbar-komponentteja. Kätevimmäksi työskentelytavaksi havaittiin Bootstrapin tyyliohjeiden noudattaminen oletuksena ja niiden ylikirjoittaminen silloin, kun se oli tarpeellista. Myös mediahakuja muokattiin sopivammille leveyksille. Itse Bootstrapin tyyliohjeisiin ei koskettu.

Responsiivisuuden saa käyttöön jakamalla ruudukkojärjestelmän kaksitoista saraketta käyttöliittymän elementtien kesken. Tämän lisäksi elementit voi jakaa riveihin, jolloin jokaisella rivillä on kaksitoista jaettavaa saraketta. Elementin paikka lasketaan sille määriteltyjen sarakkeiden määrän ja sarakkeen leveyden tulona. Jos elementti vie liikaa tilaa, se putoaa alemmas.

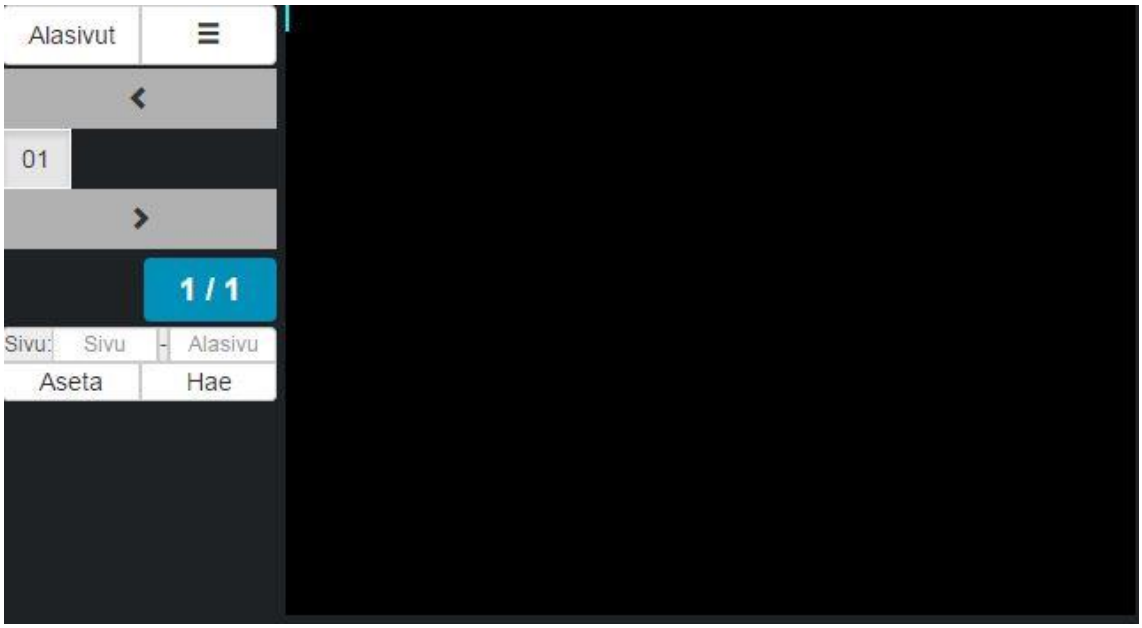
Kuva 20. Käyttöliittymän rivijako.

Työssä päädyttiin lukuisien eri käyttöliittymävaihtoehtojen jälkeen mahdollisimman yksinkertaiseen ratkaisuun (kuva 20). Käyttöliittymä jaettiin kolmeen eri riviin, joista rivit 1 ja 3 piiloutuvat erittäin pienillä laitteilla. Sarakkeita käytettiin lähinnä vain rivillä 2 leveyksien määrittämiseen, ja sarakkeiden sijasta responsiivisuus päädyttiin toteuttamaan mediahauilla. Tähän päädyttiin, koska oikealta vasemmalle alemmas siirtyvät elementit sekoittivat käyttökokemusta. Testaamalla todettiin, että mahdollisimman yhtäläinen näkymä eri laitteilla oli käytettävyydeltään paras.



Kuva 21. Käyttöliittymän responsiiviset ominaisuudet.

Kuvassa 21 on havainnollistettu käyttöliittymän responsiivisia ominaisuuksia. Suuriruutuisella laitteella käytettynä myös oikealle olevat painikkeet näkyvät. Kun näytön leveys on liian pieni niiden näyttämiseen, ne piilotetaan ja jäljelle jää vain vaaleansinisellä väritetty alue.



Kuva 22. Erittäin pieniruutuisen laitteen käyttöliittymä.

Kuvassa 22 on vastaavasti havainnollistettu, miltä käyttöliittymä näyttää erittäin pieniruutuisella laitteella. Kuvassa 21 alhaalla oleva alasivujen työkalupalkki on siirretty vasempaan reunaan ja yläreunan valikkopalkki on piilotettu oikeanpuolimmaisen painikkeen taakse.

5.11 Skaalautuvuus

Käyttäjä voi millon tahansa muuttaa selaimen suurennustasoa, joten Canvasin täytyy pystyä skaalautumaan sopivaan kokoon. Tämä on tärkeää, jotta fontti pysyy terävänä. Ilman skaalaamista fontista tulee sumea (kuva 23).

Kuva 23. Suurennettu teksti, jota ei ole skaalattu sopivan kokoiseksi.

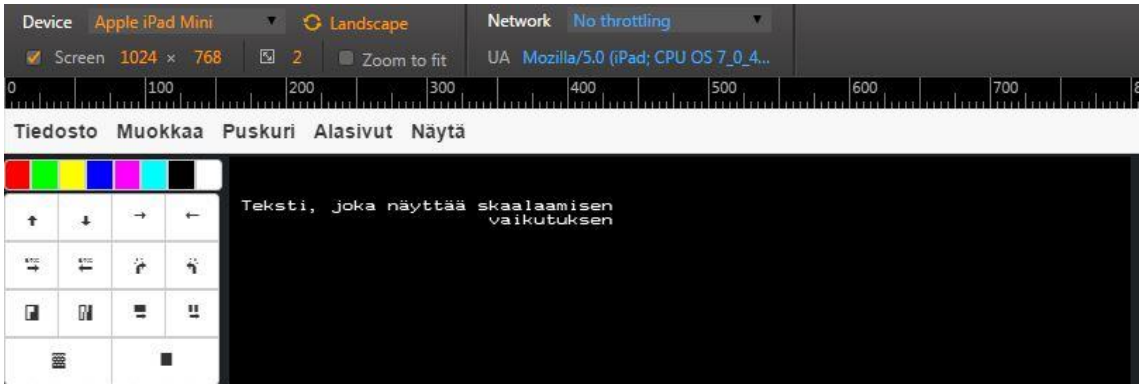
Kun Canvas skaalataan oikeaoppisesti ja dynaamisesti sopivan kokoiseksi, teksti pysyy terävänä kaikilla suurennustasoilla (kuva 24).

Kuva 24. Suurennettu teksti, joka on skaalattu sopivan kokoiseksi.

Skaalauksessa käytetään Canvasin kontekstin scale()-metodia. Parametrinä käytetään Device Pixel Ratiota, joka on käytössä olevien pikseleiden suhde näytön resoluutioon. [21.] Jos käyttäjä asettaa suurennustasoksi esimerkiksi 125 %, on DPR tällöin 1.25 eli jokaisen pikselin piirtämiseen käytetään todellisuudessa 1.25 pikseliä. Kuvat 25, 26 ja 27 havainnollistavat skaalaamisen tärkeyttä.

Kuva 25. Skaalaamisen alkutila.

Mobiililaitteissa on tyypillisesti suurempi DPR kuin 1, jotta näytön grafiikoista saadaan terävämpiä. Myös tästä syystä skaalaaminen on tärkeää, sillä muuten teksti näyttäisi liian pieneltä mobiililaitteissa (kuva 26).

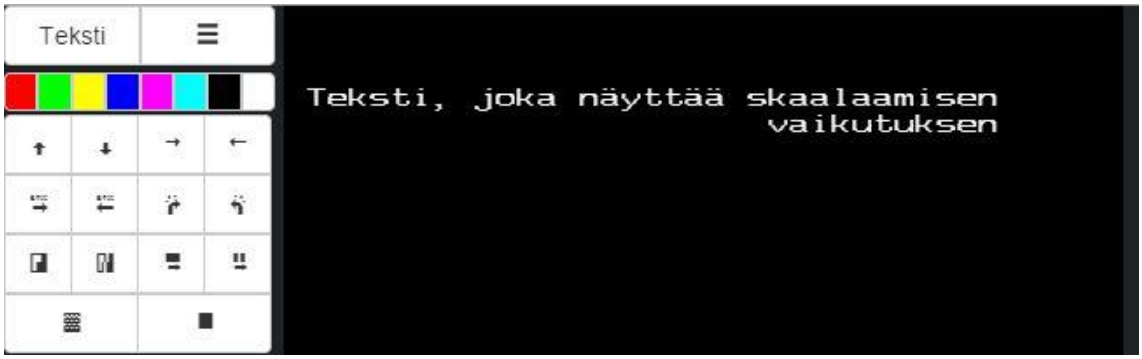


Kuva 26. Skaalaamaton Canvas-elementti mobiililaitteella emuloituna (DPR 2).

Käytännössä Canvasin käyttämien pikselien määrä kerrotaan DPR:lla, mutta Canvas piirretään näytölle CSS-tyylin leveyttä ja korkeutta noudattaen oikean kokoisena suhteessa näytön resoluutioon.

Kuva 27. Skaalattu Canvas-elementti mobiililaitteella emuloituna (DPR 2).

Lopputuloksena Canvas näyttää samalta oli päätelaitteena sitten pöytäkone tai mobiililaite ja oli suurennustaso mikä hyvänsä (kuva 27).



Kuva 28. Skaalautuminen kompaktissa käyttöliittymässä.

Canvas ja sen sisältö skaalautuvat automaattisesti oikeassa suhteessa myös responsiivisessa kompaktissa käyttöliittymässä, jossa Canvas on mittasuhteiltaan pienempi (kuva 28). Kuvasta 28 näkee myös, kuinka fontti on asetettu dynaamisesti pienemmäksi, jotta tekstin koko säilyy oikeassa suhteessa Canvasin kokoon nähden.

6 Yhteenveto

Tässä insinöörityössä esiteltiin selainpohjaisen Teksti-TV-editorin käyttämiä menetelmiä ja teknologioita, sekä joitakin editorin toteuttamisen aikana eteen tulleista mielenkiintoisista asioista ja ongelmista. Työssä esiteltiin, kuinka nykyaikainen Javascript-kehitysympäristö rakennetaan Node.js-ajoympäristöstä, paketinhallintajärjestelmästä ja avoimen lähdekoodin kirjastoista. Myös editorin kokoaminen tuotantoversioksi esiteltiin.

Työn toteutuksesta nostin esiin asioita, joita oli mielestäni hyvä esitellä tai jotka olivat mielestäni mielenkiintoisia. Teksti-TV-editori on tekstieditori, joten valmiita ratkaisuja oli jo olemassa. Tällainen löytyi rivityksen toteuttamiseen. Tekstieditorin ohjelmointipulmia oli hauska käydä läpi, koska niitä ei normaalisti tule ajatelleeksi. Esimerkiksi rivityksen saaminen toimimaan eteenpäin ja alas oli yksinkertaista, mutta taaksepäin ja ylös vaati jo enemmän pään raapimista.

Nostin myös esiin ongelman siinä, miten selaimet laskevat elementtien leveyden. Tein tämän siksi, koska mielestäni on älytöntä, että esim. textarea-elementin tekstisisältö näyttää erilaiselta eri selaimissa, vaikka se käyttäisi tasalevyistä fonttia. Editorin toteuttaminen olisi ollut paljon helpompaa, jos tällaista ongelmaa ei olisi ollut.

Editori saavutti tavoitteet mielestäni hyvin. Se toimii ja näyttää samalta suosituimmista selaimissa ja erikokoisilla laitteilla. Käytettävyys on myös samanlainen eri selainten kesken, kuten on myös sen suoritusnopeus.

Opin erittäin paljon siitä, miten Javascriptillä nykyään tehdään selainpohjaisia ohjelmia. En tuntenut ennestään muita kirjastoja jQueryn lisäksi, mutta silti kehitys lähti sujuvasti käyntiin. Yllätyin siitä, kuinka helppoa responsiivisen käyttöliittymän tekeminen oli Bootstrap-kirjastolla. Käyttöliittymän tekeminen mahdollisimman helppokäyttöiseksi ja helposti ymmärrettäväksi oli eräs omista tavoitteistani ja mielestäni saavutin sen.

Myös työn tilaaja oli kehitykseen ja lopputulokseen tyytyväinen ja miettii nyt kuinka työn tulosta voidaan hyödyntää kaupallisesti. Uskoisin, että lopputulos esitellään asiakkaalle aikanaan, korvattaessa nykyistä Java-applettia.

Lähteet

1 Matilainen, Ville. 6.10.2011. Teksti-TV pysyy ajan hermolla, välittämättä ajan hampaasta. Verkkodokumentti. <http://yle.fi/aihe/artikkeli/2011/10/06/teksti-tv-pysyy-ajan-hermolla-valittamatta-ajan-hampaasta#media=73402>. Päivitetty 7.10.2015. Luettu 6.11.2015.

2 BBC. 9.1976. Broadcast Teletext Specification. Verkkodokumentti. <http://www.bighole.nl/pub/mirror/homepage.ntlworld.com/kryten_droid/teletext/spec/teletext_spec_1974.htm>. Luettu 11.1.2016.

3 Rämö, Matti. 17.4.2014. Teksti-TV:llä edelleen miljoonayleisö. Verkkodokumentti. <http://yle.fi/tekstitv/arkisto/kalenteri/teksti-tvlla_edelleen_miljoonayleiso_9126.html>. Luettu 11.1.2016.

4 Kirchner, Lauren. 18.4.2012. Teletext Lives On In Scandinavia. Verkkodokumentti. <http://www.laurenbkirchner.com/blog/788>. Luettu 20.1.2016.

5 Git. Dokumentaatio. Verkkodokumentti. <https://git-scm.com/docs>. Luettu 21.1.2016.

6 Atlassian. Git Workflow. Verkkodokumentti. <https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow>. Luettu 27.11.2015.

7 Semantic Versioning 2.0.0. Verkkodokumentti. <http://semver.org/>. Luettu 6.11.2015.

8 Node Package Manager. Dokumentaatio. Verkkodokumentti. <https://docs.npmjs.com/misc/semver>. Luettu 7.11.2015.

9 Bower. Verkkodokumentti. <http://bower.io/>. Luettu 7.11.2015.

10 Ogden, Max. 1.2015. Nested Dependencies. Verkkodokumentti. <http://maxogden.com/nested-dependencies.html>. Luettu 7.11.2015.

11 W3C. 28.10.2014. HTML5 spesifikaatio. Verkkodokumentti. <http://www.w3.org/TR/html5/scripting-1.html#the-canvas-element>. Luettu 24.11.2015.

12 Kyrnin, Jennifer. Why Use HTML5 Canvas. Verkkodokumentti. <http://webdesign.about.com/od/html5tags/a/why-use-html5-canvas.htm>. Luettu 7.11.2015.

13 Mowery, Keaton & Shacham, Hovav. Pixel Perfect: Fingerprinting Canvas in HTML5. Verkkodokumentti. <https://cseweb.ucsd.edu/~kmowery/papers/html5-fingerprint.pdf>. Luettu 7.11.2015.

14 Backbone.js. Dokumentaatio. Verkkodokumentti. <http://backbonejs.org/>. Luettu 7.11.2015.

15 RequireJS. Dokumentaatio. Verkkodokumentti. <http://requirejs.org/docs/api.html#define>. Luettu 26.11.2015.

16 Gulp. 22.1.2014. Running Tasks in Series, i.e. Task Dependency. Verkkodokumentti. <https://github.com/gulpjs/gulp/blob/master/docs/recipes/running-tasks-in-series.md>. Päivitetty 31.8.2015. Luettu 20.1.2016.

17 W3C. 19.11.2015. HTML Canvas 2d Context. Verkkodokumentti. <http://www.w3.org/TR/2dcontext/#best-practices>. Luettu 12.1.2016.

18 Wikipedia. 13.8.2005. Line Wrap and Word Wrap. Verkkodokumentti. <https://en.wikipedia.org/wiki/Line_wrap_and_word_wrap>. Päivitetty 19.11.2015. Luettu 27.11.2015.

19 Crowley, Charles. 10.6.1998. Data Structures for Text Sequences. Verkkodokumentti. <https://www.cs.unm.edu/~crowley/papers/sds.pdf>. Luettu 20.1.2016.

20 Bootstrap. CSS. Verkkodokumentti. <http://getbootstrap.com/css/>. Luettu 1.12.2015.

21 Lewis, Paul. 25.8.2012. High DPI Canvas. Verkkodokumentti. <http://www.html5rocks.com/en/tutorials/canvas/hidpi/>. Luettu 7.11.2015.