Eric Keränen

Verokoneen uudistus

Metropolia Ammattikorkeakoulu
Insinööri (AMK)
Tieto- ja viestintätekniikka
Insinöörityö
17.11.2023

Tiivistelmä

Tekijä:	Eric Keränen
Otsikko:	Verokoneen uudistus
Sivumäärä:	39 sivua
Aika:	17.11.2023
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Ohjelmistotuotanto
Ohjaajat:	Johtava sovelluskehittäjä Miro Nurmela Lehtori Simo Silander

---

Tämä opinnäytetyö käsittelee frontend-kehitystyötä, joka toteutettiin verkkosivuston uudistamiseksi verotukseen liittyvän tiedon jakamista varten. Työssä käytettiin React-kirjastoa, joka mahdollisti nopeiden käyttöliittymien luomisen. Opinnäytetyössä käydään läpi verotietolistan uusia ominaisuuksia, mitä ne tekevät ja komponenttipohjaista ajattelua. Lisäksi tarkastellaan käyttäjäkokemuksen parantamista, joka oli yksi projektin keskeisistä tavoitteista.

Tuloksena syntyi käyttäjäystävällisempi ja informatiivisempi verotietolista, joka jaettiin Suomen suurimmilla uutisnettisivuilla nähtäväksi veropäivänä. Vanhaa toteutusta ja uudistettua toteutusta verrattiin ulkonäöllisesti. Opinnäytetyö antaa hyvän katsauksen frontend-kehityksestä yksinkertaisella nettisivulla.

Avainsanat: React, Storybook, Vite, Verokone, Helsingin Sanomat, komponenttipohjainen ajattelu, frontend

---

Abstract

Author:	Eric Keränen
Title:	Renewal of Verokone
Number of Pages:	39 pages
Date:	17 November 2023
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Professional Major:	Software Engineering
Supervisors:	Miro Nurmela, Lead Developer Simo Silander, Senior Lecturer

---

This thesis focuses on frontend development carried out to revamp a website for sharing taxation-related information. React library was used for creating fast user interfaces. The thesis discusses the new features of the tax information list, their functionalities, and a component-based approach. Additionally, it explores improving user experience, a key goal of the project.

The result is a more user-friendly and informative tax information list, showcased on major news websites in Finland on tax day. A visual comparison is provided between the old and the revamped implementations. The thesis provides a comprehensive overview of frontend development on a straightforward website.

Keywords: React, Storybook, Vite, Verokone, Helsingin Sanomat, component-based thinking, frontend

Sisällys

Lyhenteet

HTML: *Hypertext Markup Language.* HTML on merkkauskieli, jota käytetään verkkosivujen ja verkkosisältöjen luomiseen ja esittämiseen. Se toimii perustana verkkosivujen rakenteelle ja sisällölle.

TS: *TypeScript*. TypeScript on tyypitetty laajennus JavaScriptiin. TypeScirpt toimii ECMAScriptin yliluokkana.

ES5: *ECMAScript 2009.* ECMAScript on standardoitu ohjelmointikieli, joka toimii perustana esimerkiksi JavaScriptille. EcmaScript 2009 tai ES5 on ensimmäinen merkittävä uudistus JavaScriptiin.

JSX: *JavaScript XML*. JSX on syntaksilisäys JavaScriptiin, joka mahdollistaa HTML-kaltaisten elementtien käytön XML-syntaksin kautta.

XML: *Extensible Markup Language*. XML on merkkauskieli, jota käytetään tietojen tallentamiseen ja siirtämiseen tietokonejärjestelmien välillä. XML muistuttaa HTML-merkkauskieltä.

TSX: *TypeScript XML*. TSX on TypeScriptin ja JSX:n yhdistelmä, mikä mahdollistaa JavaScriptin kirjoittamisen, joka näyttää HTML-merkkauskieleltä.

HMR: *Hot Module Replacement.* Hot module replacement eli “kuuma moduulien vaihto” on kehitystyökalu, jota käytetään etenkin lokaalissa kehityksessä frontend-sovelluksissa. Se mahdollistaa koodimuutosten tekemisen ilman tarvetta päivittää koko sovellusta tai sivua.

URL: *Uniform Resource Locator*. URL on yhtenäinen resurssinpaikannin, joka määrittelee tietyn resurssin sijainnin verkkotietokoneverkossa. URL toimii myös osoitteena, jonka avulla voidaan löytää ja käyttää erilaisia verkkosivuja.

REST API: *Representational State Transfer Application Programming Interface*. REST API on ohjelmointirajapinta, joka perustuu REST-arkkitehtuuriin. REST API tarjoaa standardoidun tavan kommunikoida eri ohjelmistojen ja palvelinten välillä verkossa. REST API käyttää HTTP-protokollaa ja hyödyntää sen verkkomaisia periaatteita.

HTTP: *Hypertext Transfer Protocol*. HTTP on protokolla, jota käytetään tiedon siirtämiseen ja kommunikointiin verkon yli. HTTP mahdollistaa tiedonsiirron selaimen ja palvelimen välillä.

JSON: *JavaScript Object Notation*. JSON on kevyt tietomuoto, joka on helposti luettava ja kirjoitettava sekä ihmisille että koneille. JSON perustuu JavaScript-kielen objektirakenteisiin.

CSS: *Cascading Style Sheets*. CSS on tyylikieli, jota käytetään määrittelemään, miten HTML-dokumenttien elementit esitetään näytöllä.

RTL: *React Testing Library*. RTL on kirjasto, joka on tarkoitettu React-sovellusten testaamiseen.

DOM: *Document Object Model*. Dokumenttioliomalli on asiakirja objektimalli. DOM on ohjelmointirajapinta, joka mahdollistaa esimerkiksi HTML-dokumentin rakenteellisen edustuksen ja sen manipuloinnin käyttäen esimerkiksi JavaScriptia.

AWS: *Amazon Web Services*. Amazonin tarjoama pilvipalvelualusta. AWS tarjoaa laajan valikoiman palveluita, kuten tallennusta, tietokantoja ja muita resursseja, joita voi käyttää verkkosovellusten ja -palveluiden kehittämiseen ja ylläpitoon.

1 Johdanto

Opinnäytetyön tarkoituksena on kehittää uusi ratkaisu Verokone-nimisen verotietoja tarjoavan nettisivun upotuksen (engl. *embedding*) uudistamiseksi. Upotus on ikään kuin ikkuna toiselle nettisivulle. Kehitystyössä uudistetaan ainoastaan Verokoneen upotuksen käyttöliittymä (engl. *frontend*). Palvelinpuoli (engl. *backend*) pysyy koskemattomana. Verokone tarjoaa verotietoja omilla nettisivuillaan, mutta Verokoneen pääasiallinen tarkoitus on tarjota upotuksia Helsingin Sanomien, Ilta-Sanomien sekä Aluemedian nettisivujen web-artikkeleille.

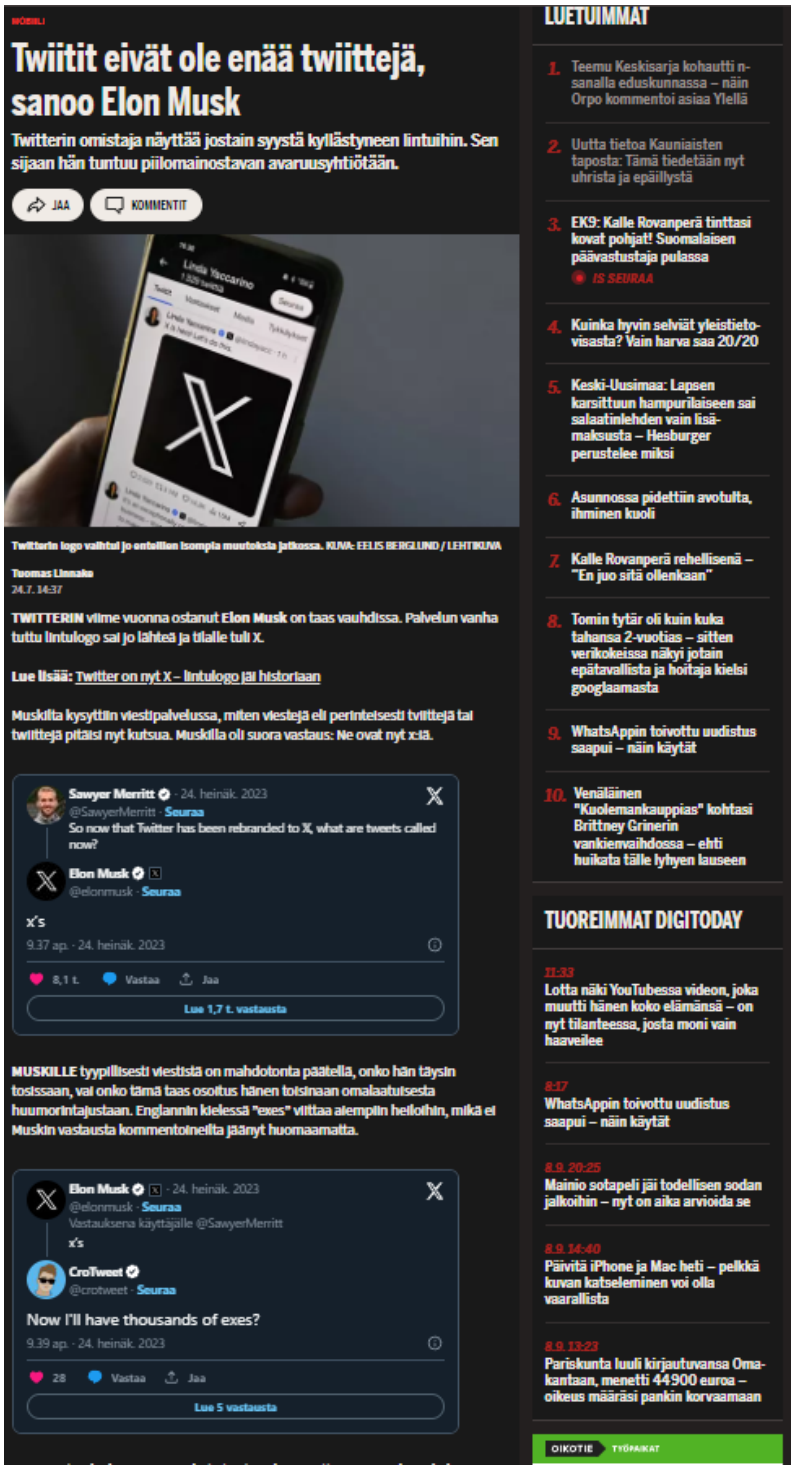
Uudistamisen tarkoituksena on siirtyä vanhoista teknologioista uusiin. Verokoneen nykyinen käyttöliittymä on tehty käyttäen Javaa ja ThymeLeaf-sapluunamoottoria (engl. *template engine*). Tämä teknologiayhdistelmä on toimiva, mutta ei sovi Verokoneen tulevaisuuteen. Verokoneesta tulee pitkällä tähtäimellä komponenttikirjasto, johon nykyinen teknologiayhdistelmä ei toimi. Uudistus tarjoaa myös ratkaisun upotusten räätälöintiin, sillä Sanoman alla olevilla brändeillä on erilaiset ulkoasut sekä toiminnalliset vaatimukset verotietojen julkaisuun.

Opinnäytetyössä käsitellään uudistamisprosessin vaiheita, kuten käyttöliittymäsuunnittelua, komponentin kehitystyötä, sisällön hallintaa ja verkkosivuston optimointia. Lisäksi tarkastellaan julkaisumenetelmää ja tapaa saada projektista syntyvä nettisivu upotettua johonkin Sanoma Median web-artikkeliin, joka käsittelee verotietoja. Lopuksi vertaillaan uuden toteutuksen ja vanhan toteutuksen nopeutta, käyttäjäystävällisyyttä, kävijämäärää sekä helppokäyttöisyyttä.

Opinnäytetyö toteutetaan Sanoman kehitystiimin jäsenenä päätoimisesti yksilötyönä. Tavoitteena on luoda uuden upotuksen suunniteltujen luonnosten pohjalta nettisivu ja upottaa se artikkeliin. Onnistuneen nettisivun upottamisen jälkeen toteutus esitellään toimittajille. Toimittajat käyttävät tätä uutta upotusta veropäivänä verotietoja käsittelevien artikkeleiden sisällä. Projektin takaraja on lokakuun loppupuolisko, sillä marraskuun 8. päivä julkaistaan vuoden 2022 verotiedot.

Tavoitteena on päivittää vanhentuneilla teknologioilla toteutettu Verokoneen artikkeliupotus käyttäen moderneja teknologioita. Työn tuloksena syntyy nettisivu, joka on integroitavissa Helsingin Sanomien, Ilta-Sanomien tai Aluemedian artikkeleihin. Työssä käsitellään pääasiassa Verokonetta, joka viittaa tässä yhteydessä nimenomaan Verokoneen artikkeliupotukseen. Verokoneella on omat erilliset nettisivunsa sekä artikkeliupotuksensa.

Upotuksella tarkoitetaan nettisivun näyttämistä toisella nettisivulla. Tämä tapahtuu nettisivuilla käytetyn HTML-merkkauskielen (engl. *Hypertext Markup Language*) tarjoaman iframe-elementin avulla. Iframe-elementti määrittelee, mikä nettisivu pyritään hakemaan iframe-elementtiä käyttävälle nettisivulle. Kuvassa 1 on havainnollistettu, mikä upotus on. Upotukset voivat olla joko staattisia tai dynaamisia samalla tavalla kuin nettisivut. Verokone on eräänlainen hybridi toteutus riippuen, kuinka upotuksen räätälöi. Verokoneella pystyy hakemaan lisää tietoa artikkelissa eli tieto muuttuu. Verokoneesta pystyy myös tekemään staattisen ottamalla suodatuksen ja etsimisvaihtoehdot pois käytöstä.

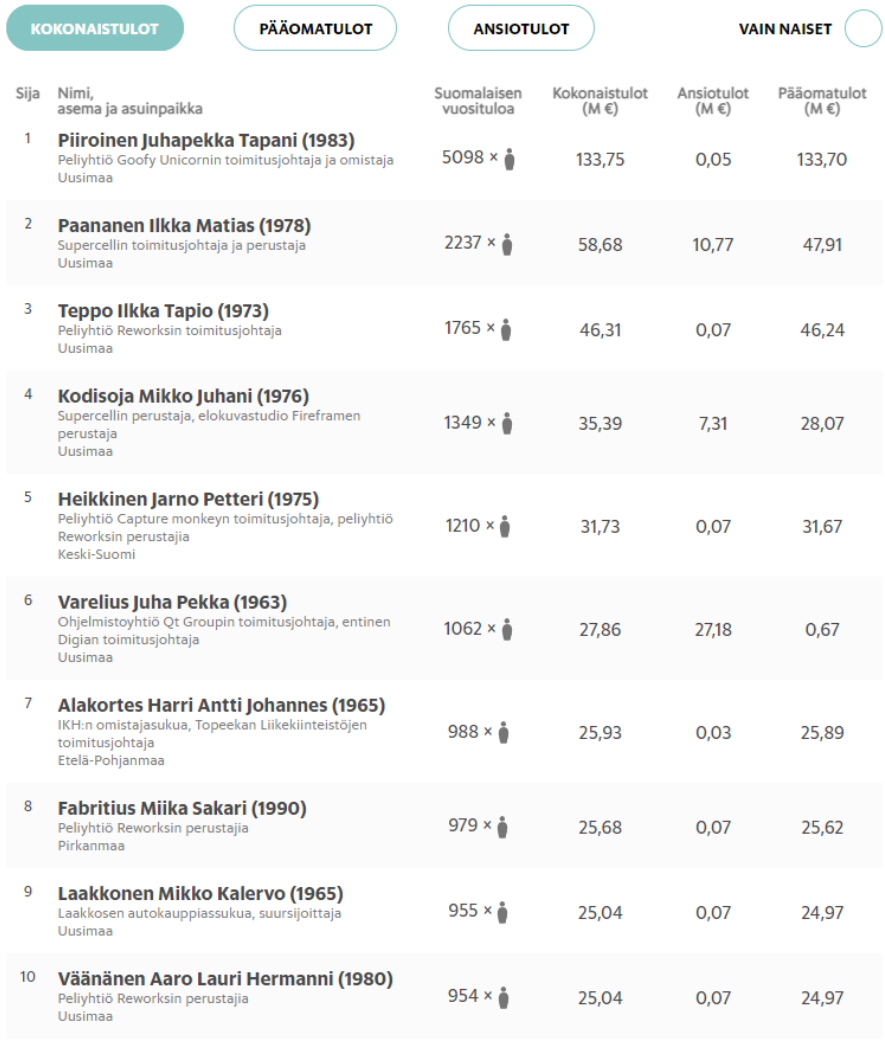


Kuva 1. Kuvankaappaus Ilta-Sanomien artikkelista, jossa on kaksi Twitter-upotusta.

Kuvassa 1 näkyvässä artikkelissa esiintyy kaksi sinisellä taustavärillä olevaa Twitter-upotusta. Twitter-upotukset haetaan Twitterin (nyk. X) omilta sivuilta ja tuodaan omina nettisivuina kyseiseen Ilta-Sanomien artikkeliin.

Verokone tarjoaa samanlaisen ratkaisun upottamalla iframe-elementtien avulla listoja veronmaksajista. Toimittajille annetaan upotuksen nettisivun osoite, josta voi luoda artikkelille sopivan verotietolistan. Toimittajien ei tarvitse tehdä muuta kuin liittää nettisivun osoite iframe-koodin kautta artikkeliin, jonka jälkeen selain hakee kyseisen nettisivun ja liittää sen artikkeliin.

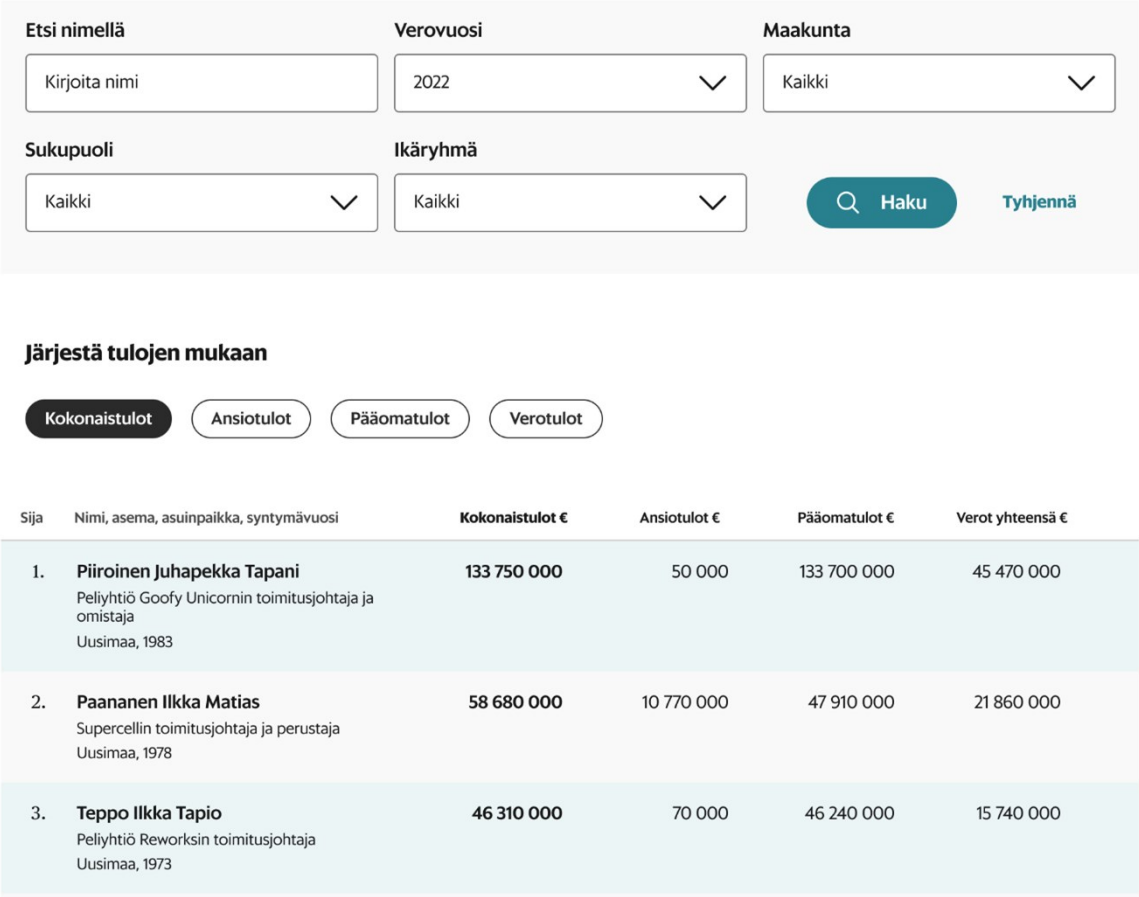
Nykyinen Verokone on tarjoiltu käyttäjille omana sivuna sekä erillisinä upotuksina artikkeleihin. Nykyisen Verokoneen tarjoaman nettisivun voi nähdä https://verokone.hs.fi/-sivulta ja artikkeliupotuksen voi nähdä kuvasta 2.



Sija	Nimi, asema ja asuinpaikka	Suomalaisen vuosituloa	Kokonaistulot (M €)	Ansiotulot (M €)	Pääomatulot (M €)
1	**Piiroinen Juhapekka Tapani (1983)** Peliyhtiö Goofy Unicornin toimitusjohtaja ja omistaja Uusimaa	5098 ×	133,75	0,05	133,70
2	**Paananen Ilkka Matias (1978)** Supercellin toimitusjohtaja ja perustaja Uusimaa	2237 ×	58,68	10,77	47,91
3	**Teppo Ilkka Tapio (1973)** Peliyhtiö Reworksin toimitusjohtaja Uusimaa	1765 ×	46,31	0,07	46,24
4	**Kodisoja Mikko Juhani (1976)** Supercellin perustaja, elokuvastudio Fireframen perustaja Uusimaa	1349 ×	35,39	7,31	28,07
5	**Heikkinen Jarno Petteri (1975)** Peliyhtiö Capture monkeyn toimitusjohtaja, peliyhtiö Reworksin perustajia Keski-Suomi	1210 ×	31,73	0,07	31,67
6	**Varelius Juha Pekka (1963)** Ohjelmistoyhtiö Qt Groupin toimitusjohtaja, entinen Digian toimitusjohtaja Uusimaa	1062 ×	27,86	27,18	0,67
7	**Alakortes Harri Antti Johannes (1965)** IKH:n omistajasukua, Topeekan Liikekiinteistöjen toimitusjohtaja Etelä-Pohjanmaa	988 ×	25,93	0,03	25,89
8	**Fabritius Miika Sakari (1990)** Peliyhtiö Reworksin perustajia Pirkanmaa	979 ×	25,68	0,07	25,62
9	**Laakkonen Mikko Kalervo (1965)** Laakkosen autokauppiassukua, suursijoittaja Uusimaa	955 ×	25,04	0,07	24,97
10	**Väänänen Aaro Lauri Hermanni (1980)** Peliyhtiö Reworksin perustajia Uusimaa	954 ×	25,04	0,07	24,97

Kuva 2. Nykyisen Verokoneen artikkeliupotusratkaisu.

Nykyisen Verokoneen artikkeliupotus on yksinkertainen lista veronmaksajia lajiteltuna kokonaistulojen mukaan. Listan ominaisuudet ovat aina samat. Uudistuksen tavoitteena on antaa Verokoneen artikkeliupotukselle modernimpi ja käyttäjäystävällisempi käyttöliittymä, jossa on hieman lisää ominaisuuksia. Verokoneen uudistuksen päätteeksi pitäisi artikkeliupotuksen olla kuvan 3 suunnitellun luonnoksen kaltainen.



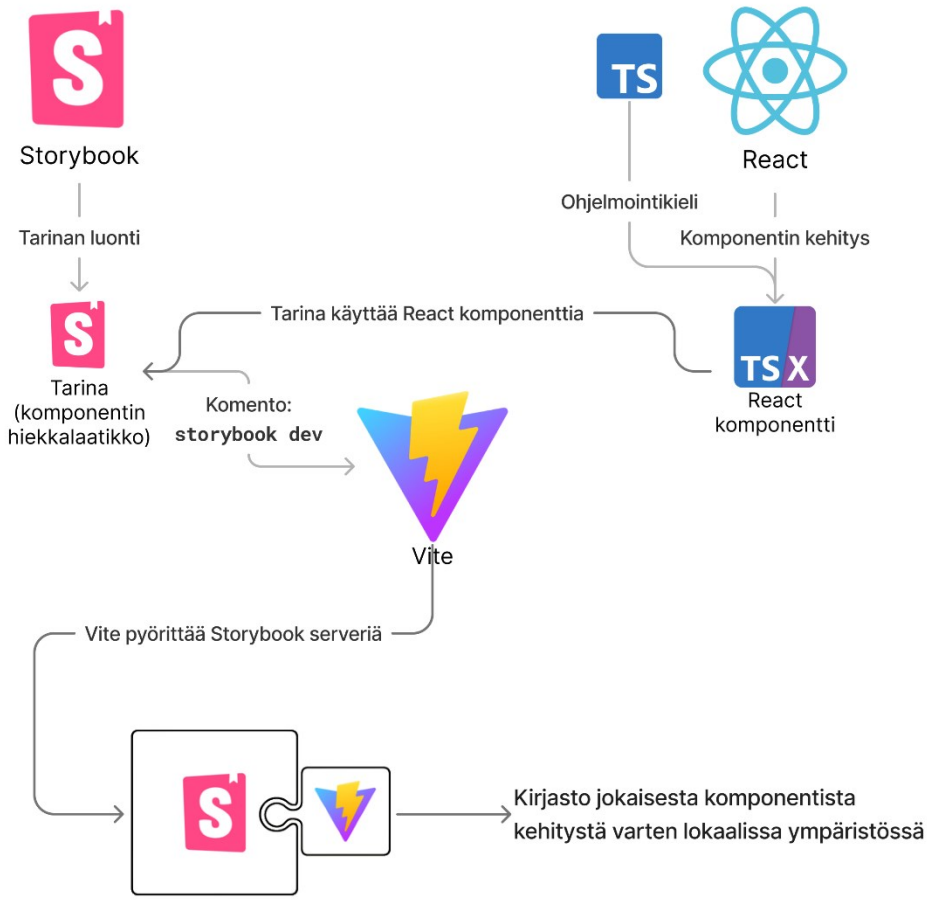
Kuva 3. Verokoneen uusi suunniteltu käyttöliittymä.

Uudistuksen ei ole tarkoitus monimutkaistaa käyttäjäkokemusta, joten miltei kaikkien ominaisuuksien kytkeminen pois päältä pitää myös olla mahdollista. Toimittajat voivat itse päättää, mitä ominaisuuksia upotuksen mukana tulee.

2 Teknologiat

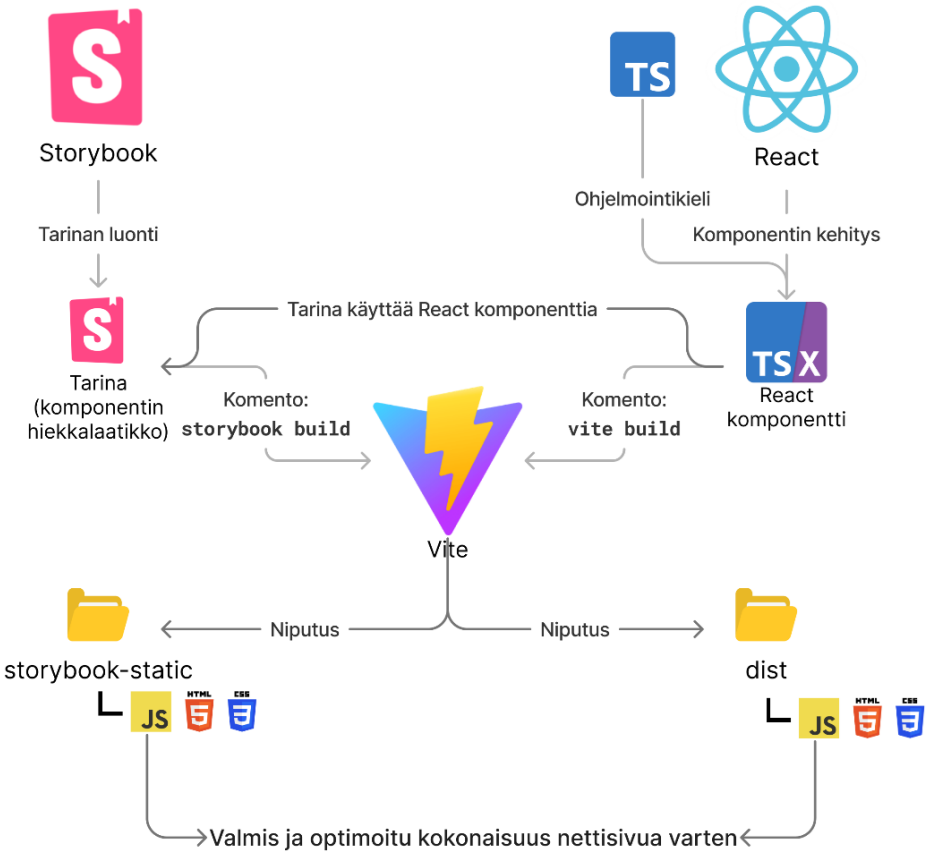
Tässä luvussa esitellään opinnäytetyöhön valitut teknologiat ja syyt teknologioiden valinnoille. Opinnäytetyön ohjelmointikieli tulee olemaan JavaScriptin päälle rakennettu TypeScript. TypeScript on valittu ohjelmointikieleksi, koska pääkirjastoksi on valittu React, joka on JavaScript -pohjainen käyttöliittymäkehitykseen luotu kirjasto. React tulee olemaan keskeisenä teknologiana koko opinnäytetyönajan ja tarjoaa hyvän tuen TypeScriptiin. Komponenttien pääasialliseen kehitystyöhön on valittu Storybook, jonka kehityspalvelimen tulee ajamaan Vite. Vite tarjoaa myös tuotantoversion niputuksen (engl. *bundling*) Storybookille ja Reactille.

Komponentit rakennetaan Reactilla ja TypeScriptillä (TS), ja niiden tarinat eli komponenttihiekkalaatikot luodaan Storybookilla. Komponenttihiekkalaatikot Storybookissa hyödyntävät Reactin komponentteja. Kehitysvaiheessa Viten ohjaama Storybook-kehityspalvelin mahdollistaa komponenttien kehittämisen komponenttihiekkalaatikoissa. Kehityspalvelimen toiminnan voi nähdä kuvasta 4.



Kuva 4. React-komponenttien kehitys Storybookin kautta. Storybook-kehityspalvelinta pyörittää Vite.

Kehitysvaiheen jälkeen, kun muutokset ovat valmiita siirrettäväksi tuotantoympäristöön, voidaan kuvan 5 avulla hahmottaa prosessi, jolla saadaan aikaan tuotantovalmiita tiedostokokonaisuuksia.



Kuva 5. Opinnäytetyön pääteknologioiden yhteydet tuotantoon mentäessä.

Tuotantoon vietäessä kuvassa 5 näkyviä komentoja käytetään komponenttihiekkalaatikoiden ja komponenttien niputtamiseen valmiiksi nettisivuksi. "storybook-static"-kansiossa on nettisivu, joka mahdollistaa kehittäjien ja suunnittelijoiden selata komponentteja yksi kerrallaan selaimessa. Jokaiselle komponentille, jolle on luotu hiekkalaatikko, on oma sivunsa. "dist"-kansiosta löytyy virallinen tuotantovalmis nettisivu, joka voidaan nettisivun osoitetta käyttämällä upottaa Sanoma Median artikkeleihin. Tuotantovalmis nettisivu on yhteen kasattu kokonaisuus kaikista kehitetyistä komponenteista, joita tarvitaan.

2.1 TypeScript

Verotietoja pitää pystyä tarkastelemaan upotuksesta ja luottamaan siihen, että data on virheetöntä. Verotietojen listaukseen kuitenkin vaaditaan joustavuutta, sillä verotiedot saattavat olla puutteellisia. Opinnäytetyön pääteknologian ollessa React edellisten vaatimusten perusteella TypeScript on oiva valinta.

Vaikka JavaScript on menestynyt ohjelmointikieli, pysyy se silti kehnona kielenä ylläpitää ja kehittää suuria ohjelmia. JavaScriptista puuttuvat luokat (engl. *classes*), rajapinnat (engl. *interfaces*) ja tyypijärjestelmä (engl. *type system*). TypeScriptin tarkoitus on tuoda JavaScriptiin nämä ominaisuudet. [2.] Esimerkki koodista 1 voi nähdä JavaScriptin validia koodia.

```
let greeting = “Welcome back, User!”;
greeting = 100;
console.log(greeting); // 100
```

Esimerkkikoodi 1. JavaScriptin syntaksi. Muuttujat (engl. *variable*) voivat olla mitä vain.

TypeScript on JavaScriptin laajennos ja pyrkii korjaamaan JavaScriptin puutteet. TypeScript toimii ECMAScript 5:n (ES5) yliluokkana (engl. *superset*), joka tarkoittaa sitä, että jokainen JavaScript-ohjelma voi toimia myös TypeScript-ohjelmana [2]. Tämä on tärkeää, sillä nettiselaimet eivät ymmärrä TypeScriptiä. TypeScript kääntää jokaisen TypeScript-tiedoston JavaScriptiin, jotta nettiselaimet ymmärtäisivät sitä. Tämä tarkoittaa sitä, että TypeScript toimii vain apusysteeminä kehittäjille eikä takaa toimivuutta [2]. Esimerkkikoodista 2 voi nähdä esimerkkikoodin 1 käännettynä TypeScriptiin.

```
let greeting: string = “Welcome back, User!”;
greeting = 100; // Type ‘number’ is not assignable to type ‘string’
console.log(greeting);
```

Esimerkkikoodi 2. TypeScriptin syntaksi. Muuttujat voi tyypittää, jolloin toisentyyppistä arvoa ei voi muuttujaan laittaa.

Vertaamalla esimerkkikoodeja 1 ja 2 voi huomata, että niiden syntaksi on sama, mutta TypeScriptissä voi määrittää muuttujille tyypin. Normaali JavaScript toimii

samalla tavalla TypeScriptissä, joten siksi on suositeltavaa käyttää tyyppejä. Esimerkkikoodi 1 suoriutuu normaalisti ja tulostaa numeron 100 konsoliin. Esimerkkikoodi 2 ei käänny (engl. *compile*), koska koodissa yritetään sijoittaa numero merkkijonotyyppiseen muuttujaan. TypeScript suojaa muuttujia virheiltä tällä tavoin.

TypeScriptillä on myös integraatio JSX-syntaksiin. JSX on JavaScriptin laajennos, joka mahdollistaa XML-merkkauskielen tyylisten (*Extensible Markup Language*) elementtien käytön JavaScript-koodissa. JSX tekee React-koodista helpommin luettavaa ja ymmärrettävää [3]. XML muistuttaa hyvin paljon HTML-merkkauskieltä, mikä helpottaa React-komponenttien kokonaisuuden hahmottamista. Esimerkkikoodista 3 nähdään TypeScript- ja JSX-syntaksia eli TSX-syntaksia funktion palauttamasta painike-elementistä.

2.2 React

Verokoneen on suunniteltu tulevaisuudessa olevan komponenttikirjasto. Tällä tavoin artikkeliupotukset voidaan toteuttaa upotuksien sijaan komponentteina. Komponenttien renderöiminen eli prosessointimenetelmä, jossa tietokone laskee lopullisen kuvan, on nopeampaa kuin iframen käyttö tiedon näyttämiseen. Komponentteja voi rakentaa myös muilla teknologioilla kuin Reactilla, mutta Reactia käytetään myös muissa Sanoman projekteissa, mikä parantaa integraatiomahdollisuuksia. Näistä syistä React valittiin pääteknologiaksi.

React on Facebookin (nyk. Meta) kehittämä avoimen lähdekoodin kirjasto, jonka päätarkoitus on luoda uudelleenkäytettäviä käyttöliittymän osia eli komponentteja. Komponentit näyttävät tietoa, joka muuttuu ajan kanssa. React ratkaisee muuttuvan tiedon ongelman päivittämällä komponentit vastaamaan ajan tasalla olevaa tietoa. [4.]

Reactin komponentit voivat viestiä toistensa kanssa propsien välityksellä. Komponentit voivat sisältää toisia komponentteja, mikä luo hierarkian komponenttien välille. Lapsikomponentit (engl. *child component*) sijaitsevat aina jonkin toisen

komponentin sisällä, kun taas komponentit, jotka sisältävät muita komponentteja, ovat isäntäkomponentteja (engl. *parent component*). Isäntäkomponentit välittävät lapsikomponenteille parametreja propsien avulla, mikä määrittää lapsikomponenttien sisällön ja funktionaalisuuden. Komponentit voivat olla samanaikaisesti isäntä- ja lapsikomponentteja. [5.]

React tarjoaa luokkakomponetteja (engl. *class component*) ja funktiokomponentteja (engl. *function component*). Luokkakomponentit ovat funktiokomponentteihin verrattuina monimutkaisempia eivätkä ole enää ylläpidettyjä [6]. Tässä työssä jatkossa, kun puhutaan komponenteista, puhutaan funktiokomponenteista. Esimerkkikoodissa 3 on nähtävillä Reactin funktiokomponentti yksinkertaisesta painikkeesta. Määrittelyssä on mukana rajapinta (engl. *interface*), nuolifunktio syntaksi sekä moduulin vienti (engl. *module export*).

```
interface ButtonProps {
     label: string;
}

export const Button = ({ label }: ButtonProps): React.ReactNode => {
     return <button>{label}</button>;
};
```

Esimerkkikoodi 3. Reactin funktiokomponentti modernilla TSX:n tarjoamalla syntaksilla koristeltuna. Funktiokomponentti palauttaa painike-elementin propseissa määritetyllä nimikemuuttujalla (engl. *label*).

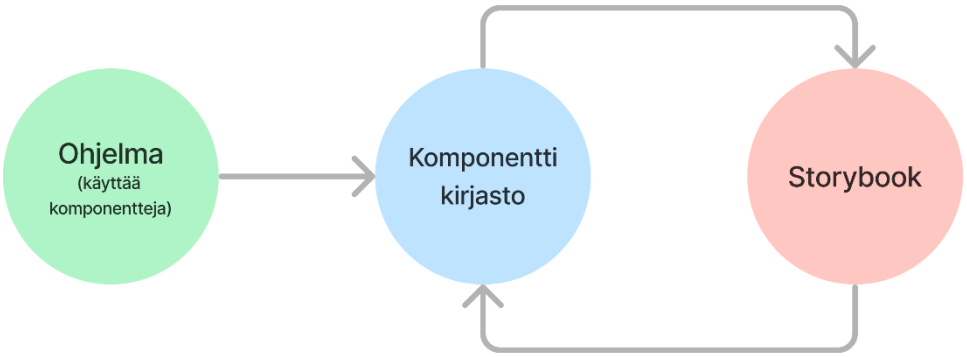
Komponenttien sisällä olevan tiedon päivittäminen ja ylläpitäminen ratkaistaan Reactin tarjoamilla ”koukuilla” eli hookeilla. Koukut ovat funktioita, jotka lisäävät ominaisuuksia ”kiinnittyen” johonkin toiminnallisuuteen, jonka jälkeen komponentti käyttää sitä käyttöliittymän näkymän renderöimiseksi käyttäjälle. Koukkuja tulee valmiiksi Reactin mukana ja myös mahdollisuus kirjoittaa omia räätälöityjä koukkuja. [7.]

2.3 Storybook

Komponenttikirjastoa kehittäessä on hyvä pitää hallinnassa kaikki tarjolla olevat komponentit jonkinlaisessa käyttöliittymässä. Suunnittelijoiden on myös hyvä

päästä kyseiseen käyttöliittymään käsiksi, jotta komponentteja voi testata ja kehittää paremmiksi. Verokoneella tulee myös olemaan paljon komponentteja, kuten kaavioita, diagrammeja ja graafisia esityksiä verotiedoista pääupotuksen lisäksi. Tätä varten työhön on valittu komponenttihiekkalaatikko nimeltä Storybook.

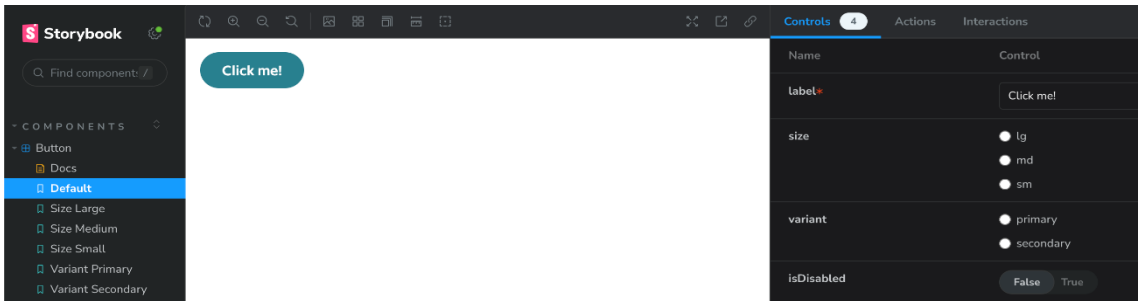
Storybook on käyttöliittymäkehitykseen tarkoitettu avoimen lähdekoodin työkalu, jolla pystyy esikatselemaan komponentteja ja muokkaamaan niiden parametreja. Storybook on eräänlainen hiekkalaatikko komponenteille ja niiden kehitykselle. Storybook toimii varinaisen pääohjelma ulkopuolella, jonka seurauksena kehittäjän ei tarvitse huolehtia riippuvuuksista ja pääohjelman vaatimuksista. Tavallisessa kehitysympäristössä tietyt komponentit voivat olla riippuvaisia tiettyihin kirjastoihin, tietoihin tai kokoonpanoihin koko sovelluksesta. Kehittäjät voivat keskittyä pelkästään yksittäiseen komponenttiin, joka on työn alla. Pääohjelman ulkopuolella toimiminen on havainnollistettu kuvassa 6. [8.]



Kuva 6. Storybook toimii pääohjelman ulkopuolella, joten ylimääräisiä riippuvuuksia tai tarkkoja ohjelmavaatimuksia ei tarvitse huomioida. Perustuu lähteeseen [8].

Storybook tarjoaa myös niin sanotun elävän dokumentaation komponenteista [8]. Storybook sisältää dokumentaation, sillä kehitysvaiheessa kirjoitetut parametridokumentaatiot ovat osa Storybookin komponenttiesityksiä. Storybook

esittää komponentit tarinoina (engl. *story*). Nämä tarinat sisältävät kaikki komponentin parametrit ja luovat parametreille vastaavat hallintaelementit niiden tyypin mukaan. Esimerkki Button-komponentin tarinasta on esitelty kuvassa 7.



Kuva 7. Storybookin näkymä. Vasemmalle on listattu Button-komponentin variaatioita. Keskellä on varsinainen komponentti. Oikealla on tarjottu komponentin parametreille kontrolleja.

Storybookilla luodaan kätevästi myös jokaisen komponentin erilaiselle muodolle ja käyttötilanteelle omat tarinat. Kuvasta 7 voi nähdä nämä erilaiset muodot vasemmalla Button-nimisen valikon alta. Ensimmäinen sivu on Button-komponentin ja sen parametrien dokumentaatio ja ensimmäisen sivun jälkeen tulee paljon variaatioita Button-komponentista. Button-komponentin parametrit määritellään komponentin propseissa Reactin puolella.

2.4 Vite

Vite on tämän opinnäytetyön viimeinen merkittävimmistä teknologioista. Se toimii paikallisen kehityksen ja kokoamistyökaluna [8]. Nämä ovat ainoat toiminnallisuudet, joita Vite tarjoaa, mutta se on erittäin optimoitu näihin kahteen tehtävään. Lisäksi Vite toimii projektin liitännäisenä, sillä sen avulla käynnistämme Storybookin paikallisen kehityspalvelimen ja kokoamme Storybookin kirjaston sekä React-sovelluksemme tuotantoa varten.

Vite tarjoaa lokaaliin kehittämiseen nopean käynnistysajan sekä yhtä nopean ominaisuuden nimeltä HMR (engl. *hot module reload*). HMR lataa lokaalissa kehitysympäristössä vain tarvittavat tiedostot lokaalin kehitysympäristön pystyttä-

miseen eikä koko projektia. Vite ohittaa myös kokonaan yli projektin pakkaamisen ennen sen tarjoamista lokaaliin ympäristöön. Kehittäessä kehittäjät voivat siis nähdä tekemänsä muutokset välittömästi nettiselaimessa lokaalissa ympäristössä, koska ylimääräistä paketointivaihetta ei ole. Vite tarjoaa myös oletuksena tuen TypeScriptille. [10.]

3 Suunnittelu ja valmistautuminen

Tässä luvussa käydään läpi Verokoneen uudistamiseen liittyvää suunnittelua ja valmistautumista.

3.1 Visio

Verokoneen uudistaminen alkoi palavereilla, jossa pyrittiin rajaamaan, mistä uudistamisessa on kyse. Tavoitteena on kirjoittaa Verokone kokonaan uusiksi moderneilla ja uusilla teknologioilla. Nykyinen Verokone on tehty käyttäen Java-pohjaista Spring-sovelluskehystä, jonka rinnalle on otettu Java-pohjainen ThymeLeaf-sapluunamoottori. Teknologiayhdistelmä on toimiva, mutta sen on koettu olevan liian vanha ja uudistuksen tarpeessa. Verokone on yksi Sanoman vanhimpia projekteja, jota ei ole vielä päivitetty.

Päivityksen myötä yhtenäistetään teknologioita, joita käytetään kaikissa Sanoman projekteissa. Samalla otetaan huomioon Verokoneen tulevat kehityssuunnitelmat. Verokoneen tulevaisuudessa siirrytään uuteen komponenttipohjaiseen frontend-ratkaisuun. Verokoneen tarjoamat listat, grafiikat, taulukot ja kaaviot muunnetaan komponenteiksi komponenttikirjaston muodossa, jotta ne sopisivat paremmin yhteen Sanoman muiden projektien kanssa.

3.2 Versionhallinta

Työtä hallinnoidaan Git-versiohallintajärjestelmän avulla. Verokoneen uudistusprojekti tallennetaan GitHubin tietovarastoon, jossa nykyinen Verokoneen toteu-

tus sijaitsee. Tällä tavoin voimme hyödyntää olemassa olevia Verokoneen resursseja. Kaikki oikeudet ja asetukset ovat myös kunnossa, joten niihin ei tarvitse käyttää ylimääräistä aikaa. Tietovarastosta löytyvät myös valmiiksi SonarCloud ja Dependabot, jotka ovat työn ylläpidon teknologioita. SonarCloud havaitsee koodivirheitä ja Dependabot päivittää riippuvuuksia ajan tasalle.

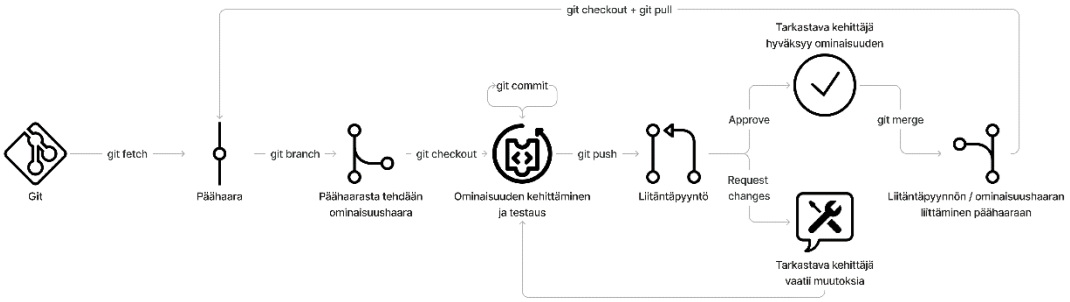
3.3 Pakettienhallintajärjestelmä

Verokonetta käytettiin testialustana myös uusille teknologioille, joita ei vielä ollut käytössä Sanoman muissa projekteissa. Yksi näistä teknologioista oli paketinhallintajärjestelmä nimeltä Yarn Berry. Yarn Berry on Yarn-paketinhallinnan toinen versio, joka kehitettiin ensimmäisen version, Yarn Classicin, jälkeen [11]. Opinnäytetyössä termi "Yarn" viittaa aina Yarn Berryhin, joka valittiin projektiin pakettienhallintajärjestelmäksi. Pakettienhallintajärjestelmä on välttämätön, jotta projektiin voi liittää muiden kehittämiä ratkaisuja, kuten edellä mainitut TypeScript, React, Storybook ja Vite. Tätä uutta Yarn-versiota ei ole vielä käytetty missään Sanoman projekteissa.

4 Kehitys

Tässä luvussa käsitellään artikkeliupotuksen kehitysprosessi Verokoneessa. Uudistusprojekti aloitettiin luomalla uusi kansio olemassa olevaan Verokoneen tietovarastoon, jossa hyödynnettiin uusinta Yarn-versiota projektin alustamiseen. Lisäksi projektiin lisättiin tarvittavia riippuvuuksia, kuten aiemmin mainitut TypeScript, React, Storybook ja Vite. Mukana olivat myös Jest:iä nopeampi Vitest-testikirjasto, Sanoman sisäinen design-tokens-paketti eri brändien tyyleihin, ESLint staattinen koodianalysaattori ja Prettier-koodinmuotoilija.

Kehitys tapahtui inkrementaalisesti askeleittain. Ominaisuudet oli havaittava ja pilkottava pienempiin helposti ymmärrettäviin kokonaisuuksiin. Tietyn ominaisuuden toteuttaminen sisälsi aina kuvassa 8 näkyvät askeleet.



Kuva 8. Jokaisen ominaisuuden lisäämisen git-työnkulku Verokoneen tietovaraston päähaaraan.

Kuvassa 8 on esitetty vaiheet ja komennot, joita noudatetaan ominaisuuden liittämiseksi tietovaraston päähaaraan:

1. Haetaan Verokoneen tietovaraston päähaara (engl. *main branch*) GitHubista.

2. Luodaan ominaisuushaara (engl. *feature branch*) tietovaraston päähaarasta.

3. Siirrytään ominaisuushaaraan päähaarasta.

4. Kehitystyön ja testauksen aikana muutokset lisätään valmisteluvaiheeseen (engl. *staging area*) pienissä erissä.

5. Kun ominaisuus on valmis, siirretään muutokset GitHubiin. Ominaisuushaarasta luodaan liitäntäpyyntö (engl. *pull request*) GitHubissa.

6. Liitäntäpyynnön tulee käydä läpi toisen kehittäjän tarkastus ja hyväksyntä, tai he voivat pyytää muutoksia ennen kuin liitäntäpyynnön voi yhdistää päähaaraan.

7. Tarvittaessa tehdään muutoksia ja liitäntäpyyntö tarkastetaan uudelleen muiden kehittäjien toimesta.

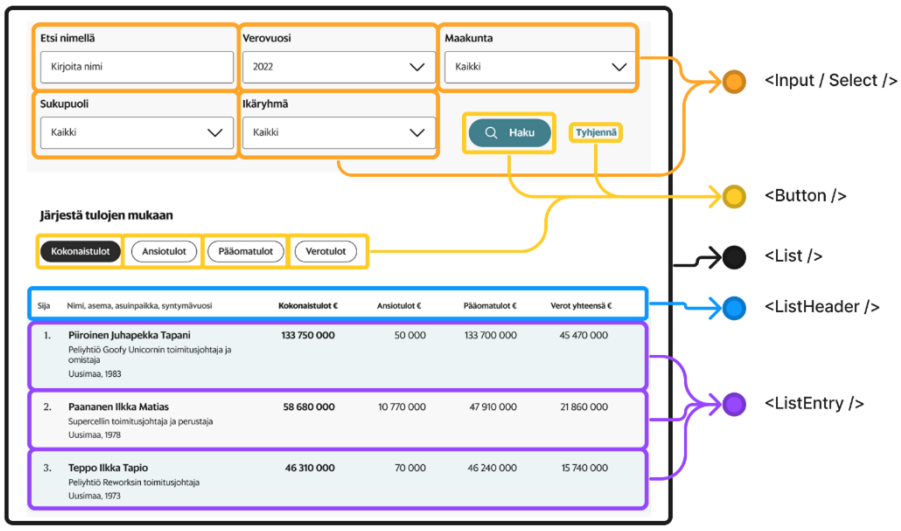
8. Hyväksytty liitäntäpyyntö yhdistetään päähaaraan.

9. Siirrytään takaisin päähaaraan ja haetaan päähaaran viimeisimmät muutokset. Viimeisimmissä muutoksissa on nyt mukana juuri liitetty ominaisuushaaran muutokset.

Verokoneen kaikkiin ominaisuuksiin, komponentteihin, muutoksiin ja korjauksiin sovellettiin nämä vaiheet. Kun ominaisuus oli valmis ja se oli integroitu tietovaraston päähaaraan liitäntäpyynnön avulla, käynnistyi automaattinen julkaisuputki, joka päivitti Sanoman sisäisen testiympäristön vastaamaan uusia muutoksia. Näin muut kehittäjät ja suunnittelijat pystyivät kokeilemaan uusia ominaisuuksia omilta koneiltaan Sanoman testiympäristössä.

4.1 Komponenttipohjaisuus

Verokoneen artikkeliupotuksen kehitys alkoi yksinkertaisella List-komponentilla. Komponenttihierarkia piti kuitenkin miettiä niin, että se olisi loogista ja sulaisi yhteen muiden komponenttien kanssa. React voi muuttaa sitä, kuinka miettii ja lähestyy ohjelmien kehitystä. Reactin tapa on pilkkoa käyttöliittymä pieniin osiin, joita kutsutaan komponenteiksi [12]. Verokoneen käyttöliittymän pilkkominen komponenteiksi on näkyvillä kuvassa 9.



Kuva 9. Verokoneen suunnitellun käyttöliittymän pilkkominen komponenteiksi.

Komponenttien pilkkominen pienempiin osiin auttaa hahmottamaan kokonaisuutta ja päättämään, mitkä komponentit on järkevää sijoittaa toisen komponentin sisään. Yleisesti haluamme, että data on lapsikomponenttien näkökulmasta lähimmällä isäntäkomponentilla, joka sisältää kyseiset lapsikomponentit. Kuvan 9 mukaan datan varastointipaikka olisi List-komponentti. List-komponentti sisältää kaikki muut komponentit, jotka liittyvät listan näyttämiseen ja sen muokkaamiseen. List-komponentin sisällä oleva tieto välitetään propsien avulla ListEntry-komponentille, joka kartoittaa (engl. *map*) tiedon. Tietojen kartoittaminen tarkoittaa sitä, että käymme läpi veronmaksajien kokoelman (engl. *array*) yksi veronmaksaja kerrallaan. Jokaisesta veronmaksajasta luodaan yksi rivi listaan, josta löytyvät veronmaksajan tiedot.

Input- ja Select-komponentteja käytetään säätämään, millä ehdoilla palvelimelta kutsutaan tietoa. Muutokset hakukenttään ja pudotusvalikoihin eivät aiheuta vaikutuksia ennen hakupainikkeen painamista. Kun hakupainiketta painetaan, se

käyttää hakukentän ja pudotusvalikoiden tietoja määrittämään, miten palvelimelta haetaan dataa. Uusi palvelimelta haettu tieto välitetään automaattisesti propsien kautta ListEntry-komponentille, joka käsittelee datan. Käytännössä Input- ja Select-komponentit vaikuttavat suoraan siihen, mitä tietoa palvelimelta haetaan.

Kuvassa 9 näkyvät neljä peräkkäistä painiketta, joilla voi järjestää tiedot tietyn tulon tai maksettujen verojen perusteella. Jokaisen veronmaksajan sijoitus määräytyy kokonaistulojen eli ansiotulojen ja pääomatulojen summan perusteella. Painikkeet tarjoavat mahdollisuuden järjestää lista kokonaistulojen, ansiotulojen, pääomatulojen tai maksettujen verojen perusteella. Listan järjestäminen tapahtuu kuitenkin palvelimen puolella, joten painikkeet tekevät kutsun palvelimelle samalla tavalla kuin hakupainike. Listan järjestäminen muulla kuin kokonaistuloilla voi hieman hämätä verotietoja tutkivaa henkilöä, sillä sijoitus määräytyy aina kokonaistulojen mukaan. Verokone ei tarjoa sijoituksia muista kuin kokonaistuloista.

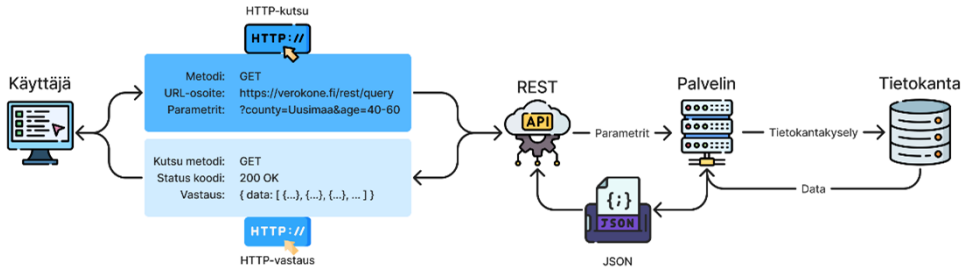
4.2 Datanhaku

Datan hakutoiminnallisuus toteutettiin käyttäen SWR-kirjastoa. SWR tarjoaa React-koukun, joka hakee dataa määritetyn noutofunktion (engl. *fetcher*) avulla URL-osoitteesta. URL (*Uniform Resource Locator*) on osoite, joka liittyy jokaiseen verkkoresurssiin [13]. Datan haku tapahtuu yhden REST API -URLin kautta, josta kaikki verotiedot haetaan. REST (*Representational State Transfer*) API (*Application Programming Interface*) eli palvelimen rajapinta vastaanottaa HTTP-kutsuja (*Hypertext Transfer Protocol*) ja välittää palvelimelle parametrit kutsun mukaisesti ja kertoo, mitä tietokannasta haetaan. Kutsussa on aina myös metodi, joka määrittelee palvelimelle, mitä halutaan tehdä. Tällainen metodi voi olla esimerkiksi:

- GET: Tietoa noudetaan
- POST: Tietoa lisätään
- PUT: Tietoa päivitetään

- DELETE: Tietoa poistetaan.

Kutsuun liitetään yleensä myös parametreja, jotka tarkentavat, mitä kutsulla halutaan [14]. Kuvassa 10 havainnollistetaan Verokoneen datanhakuprosessi, joka toteutetaan kuvitteellisen kutsun avulla. Kutsun metodi on GET ja URL-osoite on ”https://verokone.fi/rest/query?county=Uusimaa&age=40-60”. Tämä esimerkkikutsu on vain havainnollistusta varten, todellisuudessa se on erilainen.



Kuva 10. Verokoneen datanhakuprosessi.

Kuvan 10 hakuprosessi voidaan purkaa seuraavasti: Käyttäjä asettaa ensin kuvassa 9 näkyvissä olevat pudotusvalikot haluamillaan suodattimilla ja/tai hakutermillä. Tässä esimerkissä maakunta-pudotusvalikon arvo olisi ”Uusimaa” ja ikäryhmä-pudotusvalikon arvo olisi ”40-60”. Tämän jälkeen käyttäjä painaa hakupainiketta, mikä johtaa HTTP-kutsun lähettämiseen, joka on samankaltainen kuin kuvassa 10 näkyvä HTTP-kutsu. Kutsun metodi on GET, eli tietoa haetaan. URL-osoitteena on Verokoneen palvelimen verkko-osoite ja parametrit määrittyvät valittujen suodattimien ja hakutermin perusteella. Kutsu saapuu REST API:lle, joka tarkistaa kutsun tiedot. REST API toimii palvelimen osana vastaanottaen ja käsitellen kutsuja. Se siirtää hyväksytyt kutsut edelleen palvelimelle, joka muodostaa tietokantakyselyn metodista ja parametreista. Tietokanta palauttaa kyselyä vastaavat tiedot, jotka palvelin muuntaa JSON-dokumenteiksi (*JavaScript Object Notation*). JSON-dokumentit tarjoavat rakenteellisen tavan näyttää tietoa. Esimerkkikoodissa 4 on esitetty kuvitteellisen tietokannasta saadun vastauksen esimerkki JSON-muodossa. Palvelimen palauttamassa JSON-dokumentissa näkyvät parametrien mukaiset veronmaksajat.

```
{
     “offset”: 0,
     “results”: [
          {
               “name”: “Mikko Mallikas”,
               “id” “1ghj5458d232987fdds87h”,
               “county”: “Uusimaa”,
               “income”: 28000000,
               “salary”: 14000000,
               “capital”: 14000000,
               “taxes”: 8000000,
               “gender”: “Mies”,
               “birthYear”: 1977,
          },
          {
               “name”: “Essi Esimerkki”,
               ...
          },
          ...
     ]
}
```

Esimerkkikoodi 4. Verokoneen palvelimen palauttama JSON-dokumentti.

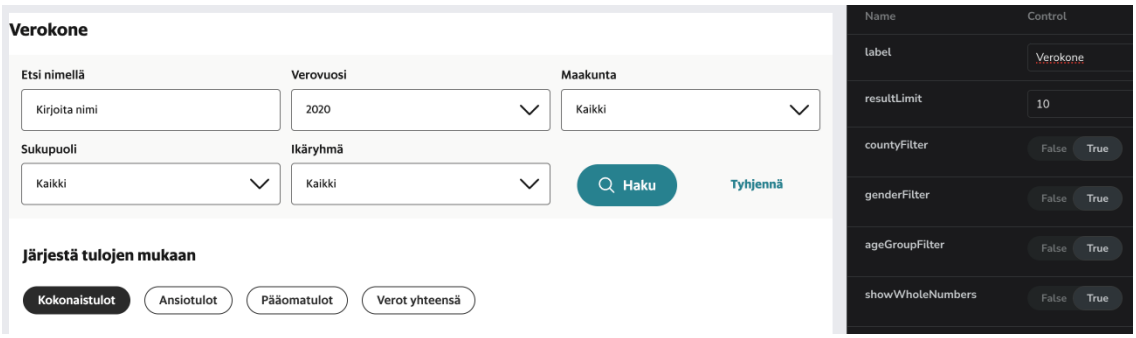
Kun vastaus on vastaanotettu, artikkeliupotukseen saadaan oikeanlaista dataa. JSON-dokumentin tiedot siirretään propsien kautta lapsikomponenteille, jotka renderöivät datan käyttäjälle nähtäväksi.

Datan hakeminen voi kestää hetken, joten on hyvä ilmoittaa käyttäjälle, että tietoja haetaan. SWR-koukku palauttaa muuttujia, jotka kertovat datan hakutilasta. Näiden muuttujien perusteella artikkeliupotus voi ilmoittaa käyttäjälle, että tietoja ladataan tai että jossain tapahtui virhe.

4.3 Parametrisointi

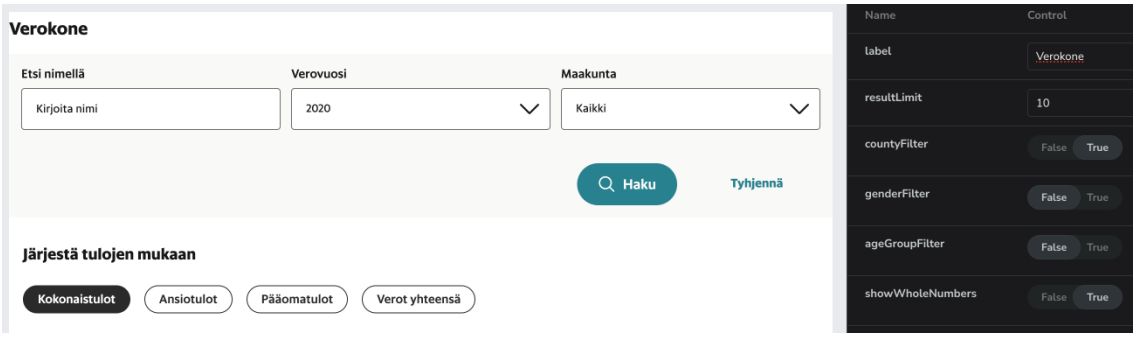
Artikkeliupotuksen yksi päävaatimus on se, että se olisi mahdollisimman joustava erilaisiin tilanteisiin. Toisinaan halutaan näyttää vain lista veronmaksajista ilman mahdollisuutta muokata listaa, kun taas toisinaan halutaan tehdä muokkauksia. Esimerkiksi artikkeli, joka käsittelee Ahvenanmaan kymmentä eniten tienannutta henkilöä, ei välttämättä tarvitse maakuntasuodatinta tai erilaisia järjestelyvaihtoehtoja. Tässä tilanteessa on hyödyllistä käyttää artikkeliupotusta, joka ennalta hakee kymmenen eniten tienannutta veronmaksajaa Ahvenanmaalta ja esittää listan sellaisenaan ilman hakumahdollisuuksia.

Lähes jokainen artikkeliupotuksen osa on parametrien takana. Jotkut näistä parametreista ovat oletusarvoisesti käytössä, kun taas toiset eivät. Artikkeliupotusta hakiessa URL:n kautta voi määritellä, mitä haluaa ottaa mukaan. Listakomponentin räätälöintiä tuotannossa URL:n kautta käsitellään tulevissa luvuissa. Parametreista on saatavilla artikkeleita kirjoittaville tahoille dokumentaatio, josta voi selvittää, mitä parametreja on käytettävissä. Kuvasta 11 näkyy esimerkki tarinasta, jossa demonstroidaan parametrien käyttöä listakomponentissa.



Kuva 11. Verokoneen listakomponentin tarina, jossa on parametreja listan suodatusta ja lajittelua varten.

Kuvasta 11 voidaan havaita, että suodattimia vastaavat arvot (countyFilter, genderFilter ja ageGroupFilter) on kaikki asetettu arvoon true. Tämä tarkoittaa, että suodattimet näytetään ja niitä voi käyttää kuten tavallisesti. Mikäli kyseessä on artikkeli, jossa esimerkiksi sukupuolen ja iän suodattimia ei haluta näyttää, koska ne on määritelty etukäteen, voidaan sukupuoli- ja ikäsuodattimet poistaa näkyvistä. Kuva 12 havainnollistaa tällaista tilannetta.



Kuva 12. Verokoneen listakomponentin tarina, jossa parametreista on otettu sukupuoli- ja ikäsuodatin pois käytöstä.

Kuvassa 12 sukupuoli- ja ikäsuodatin on poistettu näkyvistä. Tämä tarkoittaa, että näitä suodattimia ei näy eikä niitä voi muuttaa käyttöliittymässä. Listaa voi silti suodattaa maakunnan perusteella, mutta jos sukupuolisuodatin on ennalta asetettu arvoon ”Nainen”, tuloslistaan tulee vain valitun maakunnan naiset. Tällaista suodatusta voi käyttää artikkelissa, joka käsittelee yleisesti vain naisten tuloja ja veroja.

4.4 Upotuksen räätälöinti

Tässä alaluvussa käsitellään, miten upotuksen räätälöinti toteutettiin. Upotuksen räätälöinti suoritettiin URL-osoitteen avulla. Ennen upotuksen renderöintiä URL-osoite tarkistetaan, ja sieltä haetaan tarvittavat tiedot. Näiden tietojen perusteella List-komponentti alustetaan.

4.4.1 Ulkonäkö

Brändin tunnistus tapahtuu "brand"-parametrin hakemisella URL-osoitteesta. Koska Verokone on pääasiassa Helsingin Sanomien omistama tuote, oletusarvoinen brändi on aina Helsingin Sanomat. Kun brändi on tunnistettu, asetetaan tyylitys React-koukun kautta. Koukku palauttaa React-fragmentin, joka sisältää tuotemerkin mukaiset CSS-tiedostot (*Cascading Stylesheets*). CSS on yksi keskeinen tekniikka web-kehityksessä, ja se tarjoaa tyylisääntöjä HTML-elementeille. Brändin mukaiset CSS-tiedostot antavat artikkeliupotukselle brändin värimaailman, fontit ja muut tyylit.

React-fragmentit ovat tässä tilanteessa React-komponentteja ja palauttavat vain moduuleja (engl. *module imports*), joissa on halutut CSS-tiedostot. CSS-tiedostot pitää palauttaa React-fragmentteina, sillä CSS-tiedostot pitää ladata laiskasti (engl. *lazy loading*). Laiskalla lataamisella tarkoitetaan sitä, että tiedostot ladataan vasta, kun komponentti renderöidään [15]. Tällä tavoin vain tarvittavan brändin CSS-tiedostot ladataan.

Teeman tunnistaminen tapahtuu automaattisesti käyttäjän asetusten, sivuston asetusten tai mobiilisovelluksen asetusten perusteella. Teemalla viitataan joko tummaan tai vaaleaan ulkoasuun. Vaaleassa teemassa artikkeliupotuksen taustaväri on pääosin vaalea ja teksti on tummaa. Tummassa teemassa taas pääosin taustaväri on tumma ja teksti vaaleaa. Muita ulkoasuun liittyviä parametreja voivat olla esimerkiksi listaelementtien välimatkat.

4.4.2 Etsiminen

Tulosten etsiminen nimellä ja vuodella voidaan mahdollistaa "nameFilter"- ja "yearFilter"-parametreilla. Useimmissa tapauksissa Verokoneessa hakutulokset on ennakkoon määritetty ja asetettu valmiiksi "nameValue"- ja "yearValue"-parametreilla. Joissakin tilanteissa kuitenkin voi olla tarpeellista antaa käyttäjälle mahdollisuus etsiä tuloksia itse. Esimerkiksi vuosi voi olla asetettu oletuksellisesti vuoteen 2020, mutta nimellä etsiminen on sallittua käyttäjälle, jos kyseessä on artikkeli, joka käsittelee vuoden 2020 veronmaksajia.

Kaikille parametreille on mahdollista antaa oletusarvot, vaikka käyttäjälle ei annettaisi mahdollisuutta muokata suodattimia tai lajittelua. Useimmissa tilanteissa näin halutaankin tehdä, jos jokin artikkeli käsittelee tiettyjä veronmaksajia.

4.4.3 Suodatus

Verokoneessa on mahdollista tehdä suodatuksia sukupuolen, ikäryhmän ja maakunnan perusteella. Näitä suodattimia voi käyttää "genderFilter"-, "ageFilter"- ja "countyFilter"-parametrien avulla. Yleensä suodattimia käytetään ennalta määrättyinä, jolloin artikkeliupotuksessa näkyvät esimerkiksi vain Ahvenanmaan veronmaksajat. Suodattimien oletusarvot määräytyvät "genderValue"-, "ageValue"- ja "countyValue"-parametrien perusteella.

4.4.4 Lajittelu

Veronmaksajien järjestely voidaan toteuttaa ainoastaan heidän tulojensa tai maksettujen verojen perusteella. Tulolajeja ovat kokonaistulot, ansiotulot ja pääomatulot. Järjestelyn toteuttaa "sortOptionsValue"-parametri. Käyttäjä voi muokata järjestelyä vain, jos "sortOptions"-parametrin arvo on tosi.

4.4.5 Tiedon näkyvyyden hallinta

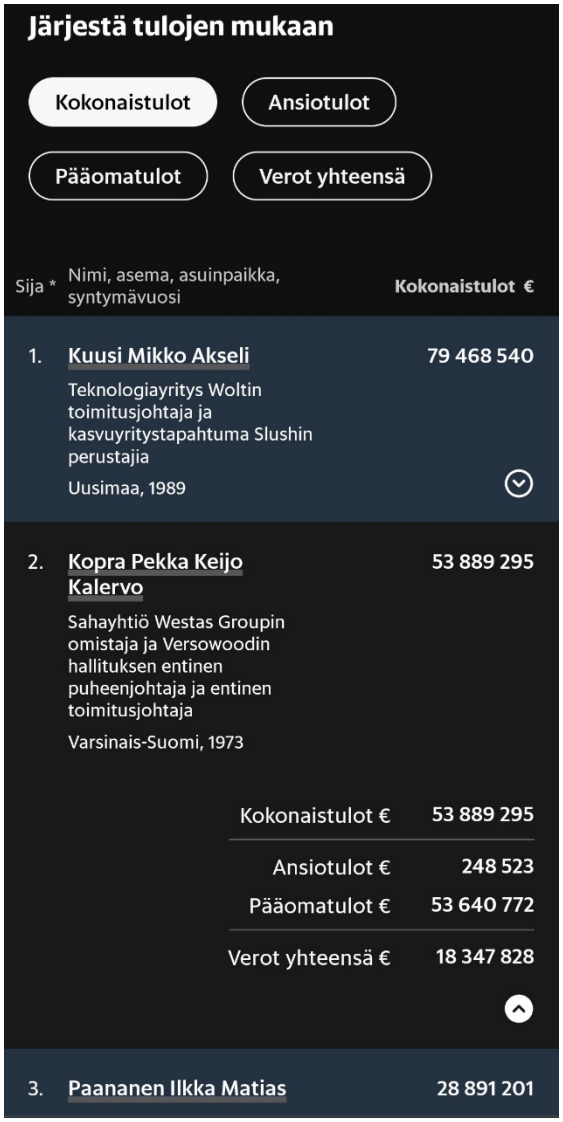
Jos jotkin tiedot ovat artikkelin kannalta merkityksettömiä, eri tiedot voidaan piilottaa upotuksessa. Näitä tietoja voivat olla esimerkiksi veronmaksajien syntymävuodet, maakunnat, tittelit ja ansaittujen tulojen muotoilu. Jotkin artikkelit saattavat olla kiinnostuneita ainoastaan veronmaksajien maakunnasta, jolloin kaikki muu tieto voidaan piilottaa lukuun ottamatta maakuntaa. Myös sijoituksen voi piilottaa kokonaan ja Helsingin Sanomissa olevat henkilöiden linkit. Helsingin Sanomissa oletuksellisesti jokainen henkilö linkitetään Verokoneen omilla sivuilla olevaan henkilösivuun.

4.5 Responsiivisuus

Sanoma Median verkkosivut ja artikkelit ovat kaikki luettavissa samalla tavalla myös mobiililaitteilla, joiden näyttökoot voivat vaihdella. Pienemmän näytön koon vuoksi kaikki verotiedot eivät välttämättä mahdu järkevästi yhdelle näytölle. Lisäksi mobiililaitteissa ei ole hiirtä, joten elementtien päällä leijuttaminen (engl. *hover*) ei toimi. Toiminnot suoritetaan painamalla näyttöä, joka vaatii mobiililaitteille räätälöidyn näkymän.

Responsiivisuuden näkökulmasta artikkeliupotus piilottaa joitakin tietoja pienillä näytöillä, jotta ensisijainen tieto on selkeästi nähtävissä käyttäjälle. Artikkeliupotuksen ennalta määrätty lajittelutapa toimii ensisijaisena tietona. Jos esimerkiksi lajittelutapana on pääomatulot, ne näytetään ensisijaisina tietoina, ja muut tulot ja verot piilotetaan pienissä näytöissä. Käyttäjällä on kuitenkin mahdollisuus painaa pientä nuolipainiketta, jolloin avautuvat kaikki muut veronmaksajan tulot ja

verot ensisijaisen tiedon alapuolella. Kuvassa 13 on näkymä Verokoneesta mobiililaitteessa.



Kuva 13. Verokoneen responsiivinen mobiilinäkymä.

Painikkeet säilyttävät normaalin kokonsa, jotta ne ovat helposti painettavissa mobiililaitteilla. Pienillä näytöillä painikkeet kasautuvat päällekkäin sitä mukaan, kun näytön koko pienenee. Responsiivisuus toteutetaan täysin CSS:n kautta.

5 Testaus ja julkaisu

Tässä luvussa käsitellään Verokoneen testausta sekä julkaisu. Koska Verokone on upotus, sen testaus tapahtuu pääasiassa Sanoman sisäisessä testiympäristössä testiartikkeleissa. Jokaisella Sanoman uutisartikkelibrändillä on oma versio niiden nettisivuista, joita käytetään pelkästään testaukseen. Kehittäjät voivat itse luoda ja kirjoittaa artikkeleita sekä kokeilla niiden toimintaa. Verokoneen tekninen puoli testataan tavallisilla yksikkötesteillä.

5.1 Yksikkötestaus

Verokoneessa jokaiselle tärkeälle komponentille on luotu yksikkötestit. Näissä testeissä käytetään Vitest-testikirjastoa yhdessä RTL:n (*React Testing Library*) kanssa. Vitest valittiin testikirjastoksi sen saumattoman yhteensopivuuden Viten kanssa, sillä molemmat teknologiat on kehittänyt sama tiimi. Vitest on myös monihaaraisuutensa ansiosta erittäin nopea, ja se pohjautuu suosittuun Jest-testikirjastoon, joka tarjoaa kattavat testaustoiminnot [16]. RTL puolestaan testaa React-komponenttien luomaa DOM-puuta ja tarjoaa lukuisia tapoja löytää oikeat elementit [17]. DOM (*Document Object Model*) on HTML-dokumenttien elementtien looginen esitystapa puurakenteena. Jokainen elementti on edustettuna objektina [18]. Esimerkkikoodista 5 voi nähdä Vitestin ja RTL:n yhteistoiminnan.

```
describe('ListEntries.tsx', () => {
     it('should render default amount of list entries', () => {
          render(<ListEntries entryList={TEST_ENTRY_LIST} />);
          except(screen.getAllByRole('row').length).toEqual(101);
     });
});
```

Esimerkkikoodi 5. List-elementin listauskomponentin testitiedosto.

Esimerkkikoodissa 5 nähdään testitiedosto, jossa käytetään Vitestin syntaksia. Testit noudattavat describe-it-except-käytäntöä [16]. Testit alkavat kontekstin luomisella (*describe*), jossa määritellään itse testi (*it*), jonka lopussa odotetaan jokin lopputulos (*except*). Testin sisällä renderöidään React-komponentti ja etsitään

kaikki riviroolin omaavat elementit (engl. *row*) DOM-puusta sekä varmistetaan, että rivielementtejä on 101 (oletuksellinen 100 veronmaksajaa ja yksi otsikkorivi).

5.2 Testaus artikkelissa

Upotuksen toimivuuden testaus tuotantotyyppisessä ympäristössä suoritettiin liittämällä upotus testiartikkeliin Sanoman sisäisessä testausympäristössä. Artikkeli luotiin Naviga-nimisellä ohjelmalla Ilta-Sanoman verkkosivujen testiympäristöön, joka vastaa tuotantoympäristöä. Artikkelin ingressiin kopioitiin iframe-elementtiin Verokoneen linkki, joka luo artikkeliin upotuksen. Verokone sijaitsee myös Sanoman sisäisessä ympäristössä, mutta eri verkkotunnuksen alla. Esimerkkikoodista 6 näkyy upotuksen HTML-merkintä (engl. *HTML markup*).

```
<iframe src="https://verokone-test.fi/article-embed?label=Verokone&re-
sultLimit=10&yearSearchValue=2022" title=”Verokone"></iframe>
```

Esimerkkikoodi 6. HTML-merkkauskielen upotuskehyksen merkitseminen.

Esimerkkikoodin 6 mukainen merkintä liitettiin Ilta-Sanomien testiartikkeliin, jotta Verokoneen upotus näkyisi artikkelissa. Upotuksen integroiminen testiartikkeliin oli merkittävä vaihe, sillä se paljasti kaksi erityspiirrettä: upotuksen korkeus ei skaalautunut oikein ja teeman vaihtaminen toiseen ei toiminut kunnolla. Nämä kaksi ongelmaa saatiin ratkaistua käyttämällä iframe-resizer-nimistä riippuvuutta, joka on valmiiksi integroitu artikkeleihin lähettämään viestejä upotuksille. Näiden viestien avulla voidaan esimerkiksi ilmoittaa upotukselle teeman muutoksesta. Oletusarvoisesti iframe-elementin korkeus lasketaan muuntamalla body-elementin marginaali pikseleiksi ja lisäämällä ylä- ja alareunojen arvot body-elementin pystysuuntaiseen sijantiin (engl. *offset height*) [19].

5.3 Kuormitustestaus

Artikkeliupotuksen kävijämäärän testaaminen toteutettiin kuormitustestaamisella. Veropäivänä artikkeliupotus tulee kokemaan kovaa liikennettä, jonka artikkeliupotuksen pitää pystyä kestämään ja ylläpitämään. Artikkeliupotuksen pitää

näkyä jokaiselle kävijälle. Kävijämäärä voi artikkeliupotuksella olla parhaimmillaan miljoonissa.

Kuormitustestaus suoritettiin Gatling-nimisellä työkalulla. Testissä käytettiin HTTP-kutsuja, joissa satunnaisesti vaihtuivat erilaiset parametrit uuden Verokoneen osalta. Yhteensä kolme testiä suoritettiin, joissa simuloitiin eri kävijämääriä eri tilanteissa. Näihin tilanteisiin kuuluivat esimerkiksi nimihaku, jossa oli eniten kävijöitä, ikähaku, maakuntahaku sekä muita mahdollisia hakuja.

Upotus kesti liikennemäärän hyvin. Ensimmäisessä kuormitustestissä testattiin Verokone ennustetulla kävijämäärällä, ja tulokset olivat hyvät. Toisessa kuormitustestissä kävijämäärä tuplattiin, mikä aiheutti koko järjestelmän kaatumisen. Verokoneinstansseja ei ollut tarpeeksi, jotta tietoa olisi voinut välittää niin monelle kutsulle. Instanssien määrää kasvatettiin ja kolmas testi tuplatulla kävijämäärällä onnistui. Testit sujuivat odotetusti, ja tuotantoon vienti hyväksyttiin.

5.4 Verokoneen julkaisu

Verokonetta varten AWS:ssä (*Amazon Web Services*) luotiin julkaisuputki (engl. *deployment pipeline*) ennen varsinaista veropäivää, jotta upotus voitiin viedä tuotantoon ajoissa. Julkaisuputki automatisoi uuden artikkeliupotuksen julkaisun Verokoneen verkkotunnukselle omassa polussaan. Verokoneen sisältöä tarjoillaan julkisesti Amazonin CloudFront-palvelun kautta.

CloudFront-palvelulla säädetään myös Verokoneen instansseja. Instansseilla määritellään, kuinka monta palvelua voi toimia samanaikaisesti, mikä mahdollistaa Verokoneen tehokkaan käsittelyn suurissa pyyntömäärissä.

Toimitukselliselle tiimille esiteltiin, miten Verokonetta voidaan käyttää artikkeleissa. Käytiin läpi eri ominaisuudet ja niiden käyttötavat. Parametrien mukainen räätälöinti Verokoneessa dokumentoitiin tarkasti ja jaettiin myös toimitukselle.

5.5 Veropäivä

Veropäivä oli 8.11.2023, ja verotustiedot saatiin verotoimistolta kello 8 aamulla. Tiedot saapuivat yksinkertaisessa CSV-muodossa (*comma-separated values*), ja ne ladattiin Verokoneeseen vuodelle 2022. Tietojen lataaminen kesti noin 7 minuuttia, jonka jälkeen ne olivat nähtävillä. Toimittajille ilmoitettiin, ja he julkaisivat artikkelinsa Verokoneen upotuksen kanssa.

Verokoneen toimintaa tarkkailtiin CloudFront-kirjauksien (engl. *log*) avulla. Kirjauksista havaittiin paljon palvelinvirheitä, jotka osoittivat, että palvelin ei toimi asianmukaisesti. Palvelin lähetti kirjauksia aikakatkaisuvirheistä (engl. *timeout*). Tästä ymmärrettiin, että tietokannasta tiedon hakemiseen tarkoitettu CloudSearch-palvelu ei pystynyt palvelemaan tulevia kutsuja tarpeeksi nopeasti. Tämä aiheutti ylikuormitusta palvelimelle, mikä johti erittäin pitkiin latausaikoihin käyttäjille tai virheilmoituksiin. Ongelmaa korjattiin lisäämällä palvelun instanssimäärää, ja virheet korjautuivat. Tämän jälkeen Verokone toimi moitteettomasti loppupäivän.

5.6 Retrospektiivi

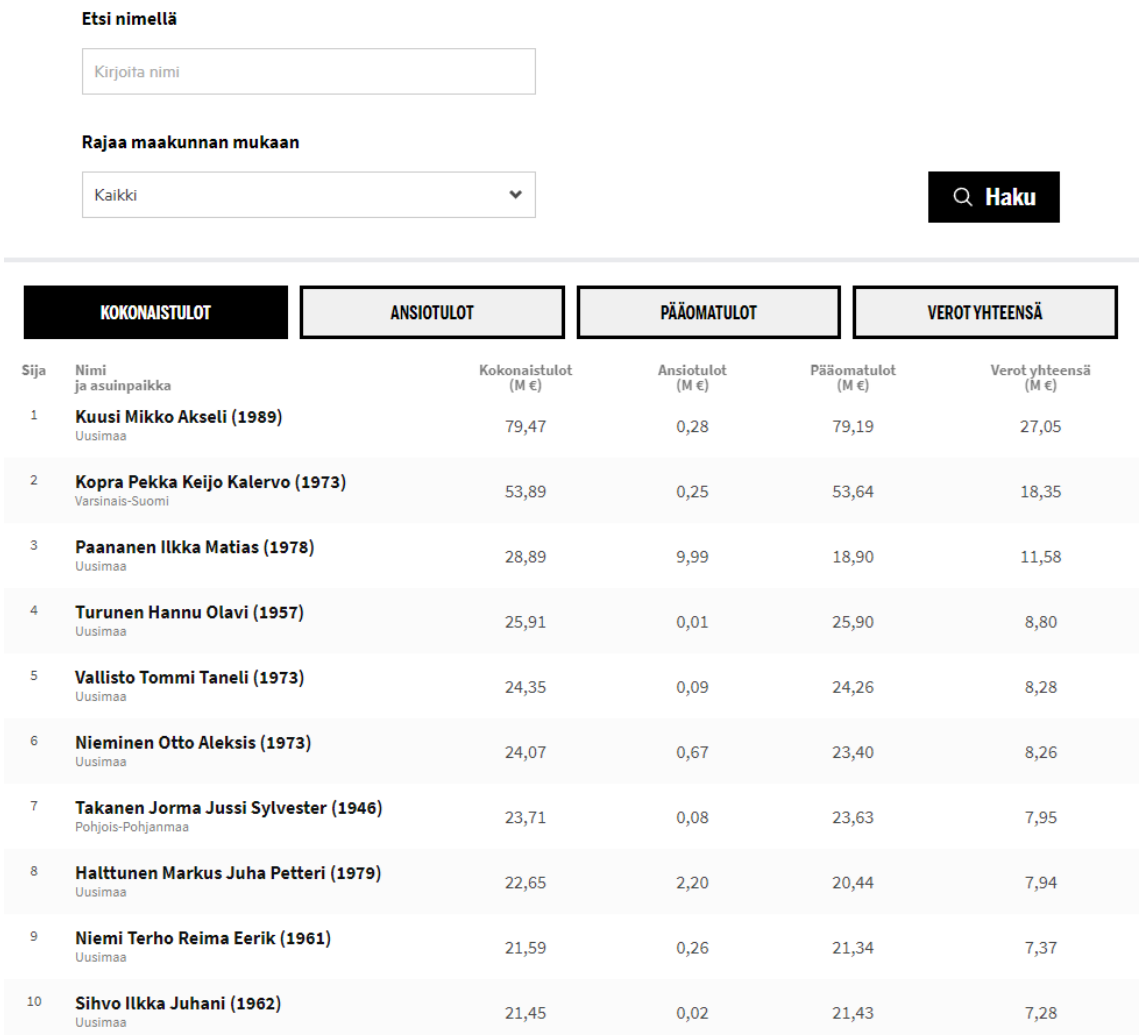
Veropäivän jälkeen aamulla ilmenneestä ylikuormituksesta laadittiin raportti. Verokoneen data kerättiin yhteen tiedostoon tutkittavaksi, jotta vastaavalta virheeltä vältyttäisiin ensi vuonna. Laadittiin tilastoraportti, jonka mukaan Verokone käsitteli Veropäivän aikana noin 112 miljoonaa kutsua, ja dataa ladattiin yli yhden teratavun verran (yli 1000 gigatavua). Yleisesti ottaen Verokonetta käytetään käytännössä vain yhden päivän vuodessa, joten instanssimäärät palautettiin takaisin normaalilukemiin. Veropäivänä veroasioita käsittelevissä artikkeleissa oli noin 6–8 miljoonaa lukijaa kirjauksien mukaan.

6 Vertailu

Tässä luvussa verrataan Verokoneen vanhaa upotustoteutusta uuteen.

6.1 Vanha upotus

Verokoneen alkuperäinen upotus on toteutettu Javaa ja Thymeleaf-sapluunamoottoria käyttäen. Tämän teknologiayhdistelmän seurauksena toteutus on täysin palvelinpuolella luotu (engl. *server-side rendered*), mikä tarkoittaa, että verkkosivu renderöidään palvelimessa eikä käyttäjän selaimessa. Kuvassa 14 näkyy alkuperäinen upotus Ilta-Sanomien tyylillä.



Etsi nimellä

Kirjoita nimi

Rajaa maakunnan mukaan

Kaikki

Haku

KOKONAISTULOT	ANSIOTULOT	PÄÄOMATULOT	VEROT YHTEENSÄ

Sija	Nimi ja asuinpaikka	Kokonaistulot (M €)	Ansiotulot (M €)	Pääomatulot (M €)	Verot yhteensä (M €)
1	**Kuusi Mikko Akseli (1989)** Uusimaa	79,47	0,28	79,19	27,05
2	**Kopra Pekka Keijo Kalervo (1973)** Varsinais-Suomi	53,89	0,25	53,64	18,35
3	**Paananen Ilkka Matias (1978)** Uusimaa	28,89	9,99	18,90	11,58
4	**Turunen Hannu Olavi (1957)** Uusimaa	25,91	0,01	25,90	8,80
5	**Vallisto Tommi Taneli (1973)** Uusimaa	24,35	0,09	24,26	8,28
6	**Nieminen Otto Aleksis (1973)** Uusimaa	24,07	0,67	23,40	8,26
7	**Takanen Jorma Jussi Sylvester (1946)** Pohjois-Pohjanmaa	23,71	0,08	23,63	7,95
8	**Halttunen Markus Juha Petteri (1979)** Uusimaa	22,65	2,20	20,44	7,94
9	**Niemi Terho Reima Eerik (1961)** Uusimaa	21,59	0,26	21,34	7,37
10	**Sihvo Ilkka Juhani (1962)** Uusimaa	21,45	0,02	21,43	7,28

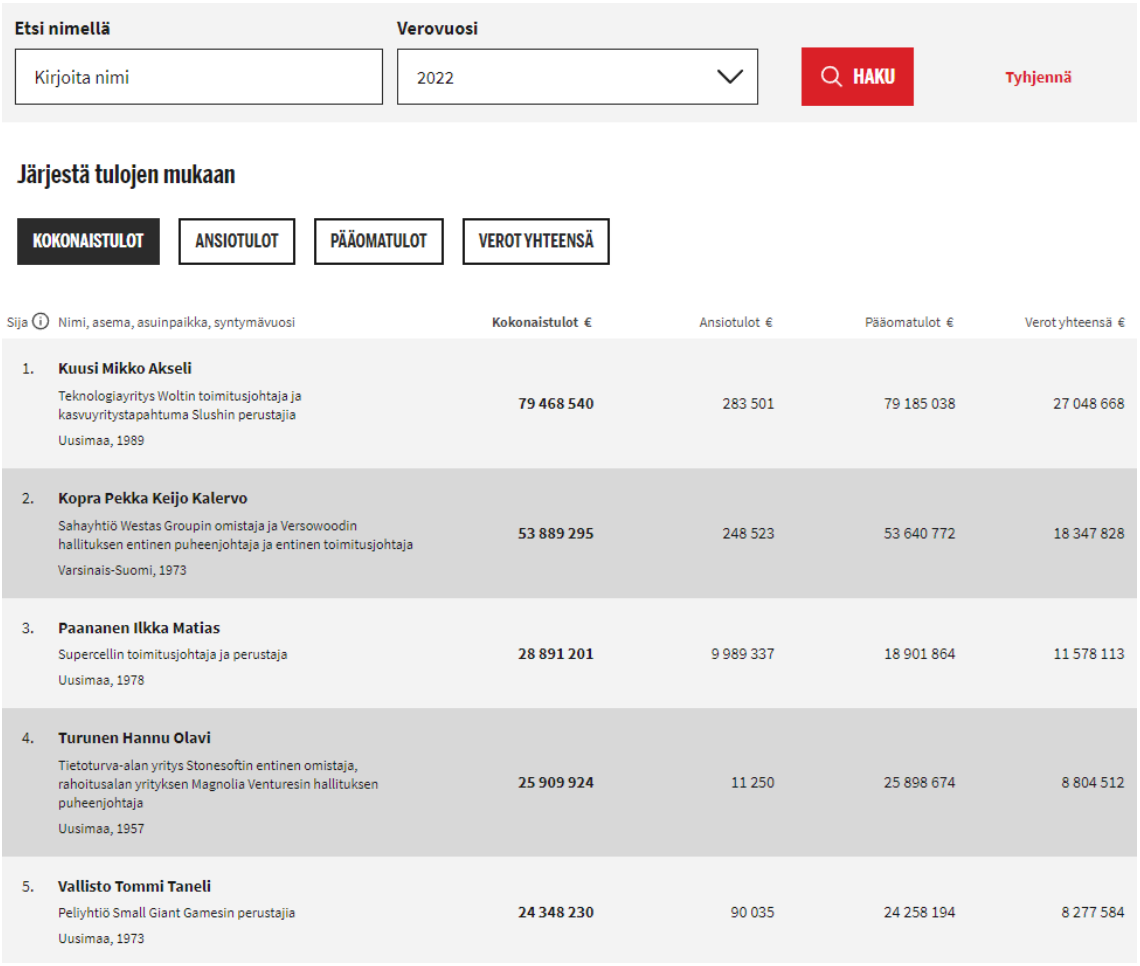
Kuva 14. Vanha Verokoneen upotus Ilta-Sanomien tyyleillä. Upotus löytyy osoitteesta https://verokone.hs.fi/iswidget

Kuvassa 14 nähtävä upotus säilyy aina samana eikä ole muokattavissa. Tyylisuunnan vaihtaminen edellyttää siirtymistä kokonaan eri sivulle, koska kyseinen

sivu on omistettu yksinomaan Ilta-Sanomien tyylille. Upotus on sivutettu, mikä tarkoittaa, että upotuksen lopussa on painike, joka ohjaa uudelle sivulle, missä on lisää tuloksia. Sivutettu toteutus vanhasta upotuksesta voi aiheuttaa hämmennystä, sillä vaihdettaessa sivua toiseen ja sen jälkeen muutettaessa lajittelua käyttäjä pysyy samalla sivulla eikä pala uuden lajittelun ensimmäiselle sivulle. Lajittelunapin painaminen muuttaa listan järjestystä uudelleen. Upotus on responsiivinen ja tarjoaa mobiilinäkymässä vain valitun suodattimen tiedot.

6.2 Uusi upotus

Verokoneen uusi upotus on rakennettu React-kirjaston ja TypeScriptin avulla. Tämän teknologian yhdistelmän ansiosta toteutus tapahtuu selaimessa (engl. *client-side rendered*). Toisin sanoen vain HTML-tiedosto luodaan palvelimella, mutta muu sisältö syntyy vasta käyttäjän nettiselaimessa. Kuvassa 15 näkyy uusi upotus Ilta-Sanomien tyylillä.



Kuva 15. Uusi Verokoneen upotus Ilta-Sanomien tyyleillä. Upotus löytyy osoitteesta https://verokone.hs.fi/embed/index.html?brand=IS

Kuvassa 15 esitetty upotus ei ole aina samanlainen, sillä URL-parametrien avulla suodattimia, henkilötietoja ja muita elementtejä voi piilottaa tai näyttää. Arvoja voi ennalta määrittää. Upotus voi palvella muita tyylejä kuin Ilta-Sanomien URL-parametrien avulla. Sivutusta ei ole; sen sijaan tuloksia ladataan jatkuvasti (engl. *infinitely scrolling*) samalle sivulle, kunnes viimeinen tulos saavutetaan. Jatkuva lataus pitää käynnistää listan lopussa olevalla painikkeella. Lajittelupainikkeella lista voidaan järjestää uudelleen. Upotus on responsiivinen ja tarjoaa mobiilinäkymässä mahdollisuuden nähdä muut tulot nuolipainiketta painamalla.

6.3 Lopputulokset

Vertailun lopputulokset voi nähdä taulukosta 1.

Taulukko 1. Verokoneupotuksien vertailutulokset.

Ominaisuus	**Vanha toteutus**	**Uusi toteutus**
Renderöinti	Palvelimessa	Nettiselaimessa
Muokattavuus	Staattinen	Muokattavissa URL-parametrien kautta
Tyylitys	Staattinen	Muokattavissa URL-parametrien kautta
Lisätuloksien haku	Sivutettu	Jatkuva vieritys
Suodatus	Suodatetaan palvelimen puolella	Suodatetaan palvelimen puolella
Lajittelu	Lajitellaan palvelimen puolella	Lajitellaan palvelimen puolella
Responsiivisuus	Responsiivinen	Responsiivinen
Informatiivisuus	Vähemmän infoa	Enemmän infoa

Vertailun keskeinen ero on upotusten renderöintitavassa. Nettiselaimessa luotu upotus näyttää aluksi aina tyhjän sivun ja lataa sisällön JavaScriptin avulla, mikä tarjoaa nopeat vuorovaikutukset mutta mahdollisesti hitaamman alkulatausajan. Palvelimessa luotu upotus puolestaan näyttää aluksi täysin renderöidyn HTML-sivun tarjoten nopean näkyvyyden, mutta saattaa kohdata haasteita suuren palvelimen kuormituksen alla. [20.]

Yhtä selvää parempaa valintaa ei ole, sillä kumpikin toteutus toimii omalla tavallaan. Vertailussa uusi upotus kuitenkin tarjoaa paljon enemmän vaihtoehtoja siihen, kuinka upotusta voi räätälöidä ja kuinka informatiivinen se on. Lisätulosten haku ei myöskään ole sekava uudessa toteutuksessa.

7 Jatkokehitys ja yhteenveto

Tämän opinnäytetyön ratkaisuna oli artikkeliupotus, joka tarjosi yksittäiselle komponentille ratkaisun. Tällainen toteutus oli helppo implementoida, koska koko ohjelma tarjosi tietoa vain yhdelle komponentille. Ratkaisut oli suunniteltu nimenomaan verotietolistalle. Jos komponentteja olisi ollut useampia kuin yksi, ohjelmareitittimen (engl. *app router*) lisääminen projektiin olisi ollut tarpeellista. Ohjelmareitittimellä voidaan luoda useita alasivuja, joissa URL-parametreja voidaan lukea eri tavoin, koska eri komponenteilla on erilaiset vaatimukset ja huomioitavat asiat parametreissa.

Tässä opinnäytetyössä luotu ratkaisu hallitsee komponenttia ja sen tarjoilua samalla ohjelmalla. Kuitenkin tulevaisuudessa komponentit siirretään omaan erilliseen kokonaisuuteensa, joka paketoidaan ja jaetaan kokonaisuutena. Muut projektit voivat sitten ottaa tämän paketin käyttöön riippuvuutena (engl. *dependency*). Pakettiin tuodaan kaikki Verokoneen komponentit (engl. *import*). Tämä paketti toimii pienenä komponenttikirjastona.

Komponentin tarjoiluun voi käyttää esimerkiksi Next.js-sovelluskehystä luomaan verkkosivun, joka tarjoilee paketoidut komponentit riippuvuuden kautta. Next.js tarjoaa kätevän ohjelmareitittimen, jonka avulla voi helposti luoda alasivuja veronmaksajien yksilöivien tunnusten perusteella [21]. Verokoneessa jokaisen veronmaksajan henkilökohtaiset tiedot talletetaan henkilön yksilöivän tunnuksen avulla. Kullekin yksilöivälle tunnukselle voidaan luoda sivu, joka sisältää veronmaksajan tiedot ja verohistorian. Pääsivulle voidaan sitten sijoittaa muut Verokoneen tarjoamat komponentit, kuten verolista, graafit, tilastot ja diagrammit.

Opinnäytetyön tavoitteena oli päivittää Verokoneen artikkeliupotus moderneilla teknologioilla ennen vuoden 2023 veropäivää. Uudistuksen keskeisenä ominaisuutena oli parametrisointi, joka mahdollistaa artikkeliupotuksen räätälöinnin eri artikkeleille sopivaksi. Uusi artikkeliratkaisu korvasi aikaisemmat brändikohtaiset artikkeliupotukset.

Verokoneen artikkeliupotus uudistettiin Reactin avulla Verokoneen tulevaisuuden suunnitelmien mukaisesti. Verokoneen tavoitteena on kehittyä komponenttikirjastoksi, ja artikkeliupotuksen päivitys oli ensimmäinen askel tähän suuntaan. Lisäksi Verokone toimi alustana uusille teknologioille, joita ei ollut käytössä muissa Sanoman projekteissa.

Uudistuksen myötä käyttöliittymää tehostettiin entistäkin intuitiivisemmaksi. Se tarjoaa verotietoja tarkastelevalle käyttäjälle enemmän tietoa sekä mahdollisuuden tehokkaaseen suodattamiseen ja listan lajitteluun. Erityisesti Verokoneen mobiilinäkymä on nyt suunniteltu entistä käyttäjäystävällisemmäksi. Parametrisoinnin avulla voi eliminoida tarpeettoman tiedon, joka ei ole olennainen artikkelille. Tämä mahdollistaa artikkeliupotuksen joustavamman käytön erilaisissa tilanteissa.

Opinnäytetyön asettamat tavoitteet saavutettiin, ja päivitetty artikkeliupotus oli toimittajien käytössä ennen veropäivää. Toimittajille esiteltiin uusi ratkaisu, ja heitä opastettiin sen käytössä, jotta he voisivat mukauttaa verolistat sopiviksi omiin artikkeleihinsa. Tämä ratkaisu yhdisti artikkeliupotuksen yhdeksi kokonaisuudeksi, minkä seurauksena muita verotietolistaratkaisuja ei enää tarvita. Aiemmin oli luotu lukuisia erilaisiin tilanteisiin soveltuvia verotietolistoja, mutta räätälöinnin ansiosta tarvitaan enää vain yksi upotus.

Veropäivänä ilmeni joitain haasteita upotuksen kanssa, mutta ne saatiin korjattua. Veropäivä oli kokonaisuudessaan merkittävä menestys, ja se toi runsaasti lukijoita Ilta-Sanomille, Helsingin Sanomille ja Aluemedian brändeille. Verokoneen artikkeleita luettiin noin 6–8 miljoonaa kertaa.

Vanhan ja uuden toteuksen vertailun perusteella havaittiin, ettei ole selkeää eroa resurssien, nopeuden ja toiminnallisuuden suhteen. Uusi upotus kuitenkin erottuu merkittävästi intuitiivisuudessa, sillä se mahdollistaa upotuksen räätälöinnin artikkelikohtaisesti. Lisäksi uusi toteutus on huomattavasti informatiivisempi kuin aikaisempi versio.

Lähteet

1 Lionel Sujay Vailshery.
Most widely utilized programming languages among developers worldwide 2023.
Verkkoaineisto.
< https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/ >
Luettu 10.9.2023.

2 Gavin Bierman, Martin Abadi & Mads Torgersen.
Understanding TypeScript.
Verkkoaineisto.
< https://users.soe.ucsc.edu/~abadi/Papers/FTS-submitted.pdf >
Luettu 10.9.2023.

3 TypeScript JSX dokumentaatio.
Verkkoaineisto.
< https://www.typescriptlang.org/docs/handbook/jsx.html >
Luettu 8.10.2023.

4 Prateek Rawat & Archana N. Mahajan.
ReactJS: A Modern Web Development Framework.
Verkkoaineisto.
< https://ijisrt.com/assets/upload/files/IJISRT20NOV485.pdf >
Luettu 10.9.2023.

5 Iiva Jorge.
Understanding React Parent Components: Best Practices and Common Techniques.
Verkkoaineisto.
< https://medium.com/@livajorge7/introduction-9e84391f4b83 >
Luettu 10.9.2023.

6 Reactin komponentti dokumentaatio.
Verkkoaineisto.
< https://react.dev/reference/react/Component >
Luettu 10.9.2023.

7 Luojus Tuomas.
Usability and Adaptation of React Hooks.
Verkkoaineisto.
< https://trepo.tuni.fi/bitstream/handle/10024/130986/LuojusTuomas.pdf?sequence=2&isAllowed=y >
Luettu 10.9.2023.

8 Aparna.
What is Storybook and Why Developers Should Use It.
Verkkoaineisto.
< https://stackify.com/what-is-storybook-and-why-developers-should-use-

it/ >
Luettu 10.9.2023.

9 Noble Okafor.
Getting Started With Vite: The Ultimate Build Tool.
Verkkoaineisto.
< https://www.makeuseof.com/vite-ultimate-build-tool-getting-started/ >
Luettu 10.9.2023.

10 Shuaina Wang, Hang Yin & Xiangkun Wang.
Patien-roeinted online healthcare services platform: development based on Vue3 and Ts.
Verkkoaineisto.
< https://drpress.org/ojs/index.php/fcis/article/view/7570 >
Luettu 10.9.2023.

11 Maël Nison (arcanis).
GitHub kommenttiketju Yarnin versioinnista.
GitHub kommenttiketju.
< https://github.com/yarnpkg/berry/discussions/3385#discussioncomment-1277876 >
Luettu 17.9.2023.

12 Thinking in React dokumentaatio.
Verkkoaineisto.
< https://react.dev/learn/thinking-in-react >
Luettu 10.9.2023.

13 URL-osoite dokumentaatio.
Verkkoaineisto.
< https://developer.mozilla.org/en-US/docs/Learn/Common_questions/Web_mechanics/What_is_a_URL >
Luettu 17.9.2023.

14 REST APIn määritelmä.
Verkkoaineisto.
< https://www.ibm.com/topics/rest-apis >
Luettu 17.9.2023.

15 What is lazy loading?
Verkkoaineisto.
< https://www.cloudflare.com/learning/performance/what-is-lazy-loading/ >
Luettu 10.10.2023.

16 Victoria Lo.
Vitest: Blazing Fast Unit Test Framework.
Verkkoaineisto.
< https://lo-victoria.com/vitest-blazing-fast-unit-test-framework >
Luettu 10.10.2023.

17 Scottie Crump.
Simplify Testing with React Testing Library.
Verkkoaineisto.
< https://books.google.fi/books?hl=en&lr=&id=XDgqEAAAQ-BAJ&oi=fnd&pg=PP1&dq=React+Testing+Library&ots=cvnBPobzDd&sig=8TU21l59SqmBEnk1eZr5ol93dfw&redir_esc=y#v=onepage&q=React%20Testing%20Library&f=false >
Luettu 10.10.2023.

18 Document Object Model dokumentaatio.
Verkkoaineisto.
< https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model >
Luettu 10.10.2023.

19 David J. Bradshaw.
iframe-resizer dokumentaatio.
Verkkoaineisto.
< https://github.com/davidjbradshaw/iframe-resizer/blob/master/docs/parent_page/options.md >
Luettu 29.9.2023.

20 Dan Taylor.
Client-Side Vs. Server-Side Rendering.
Verkkoaineisto.
< https://www.searchenginejournal.com/client-side-vs-server-side/482574/ >
Luettu 11.11.2023

21 Next.js dokumentaatio reitityksestä.
Verkkoaineisto.
< https://nextjs.org/docs/app/building-your-application/routing/pages-and-layouts >
Luettu 10.10.2023.