Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

JESSE ROMO

Angular Server Side Rendering

Ja Reppi.fi palvelun päivittäminen

TIETOJENKÄSITTELYN TUTKINTO-OHJELMA
2024

TIIVISTELMÄ

Romo Jesse: Angular Server Side Rendering ja Reppi.fi palvelun päivittäminen
Opinnäytetyö, AMK
Tradenomi, tietojenkäsittely
Maaliskuu 2024
Sivumäärä: 34

Tässä opinnäytetyössä tutustuttiin prosessiin, jossa olemassa oleva yksisivusovellus muutetaan käyttämään selainpuolen renderöintiä. Tämän prosessin keskiössä oli myös verkkosivujen statistiikan mittaaminen ja optimointi.

Työssä tutkittiin yleismallisesti Serverless kehittämistä ja sitä, miten yksisivusovellus voisi toimia Serverless-mallilla. Vertailussa oli myös yksisivusovellukset ja monisivusovellukset keskenään.

Opinnäytetyö on jaettu karkeasti kahteen pääosioon. Ensimmäisessä osiossa päivitetään Reppi.fi palvelun etusivu ja samalla muutetaan se käyttämään selainpuolen renderöintiä. Toisessa osassa tutustutaan projektissa käytettyihin teknologioihin sekä perehdytään verkkosivun statistiikan mittaukseen. Lopuksi käydään läpi kyseisiä mittaustuloksia vanhan ja uuden etusivun välillä.

Avainsanat: Angular, SPA, MPA, Single-page application, Multi-page application, Serverless, Selainpuolen renderöinti

Abstract

Romo, Jesse: Angular Server-side Rendering and updating Reppi.fi service
Bachelor's thesis
Business administration
March 2024
Number of pages: 34

In this thesis, we explore the process of converting an existing single-page application to utilize client-side rendering. At the core of this process is also the measurement and optimization of core web vitals.

The thesis investigates Serverless development in general in the context of single-page applications and how a single-page application could operate within the Serverless model. Additionally, there is a comparison between single-page applications and multi-page applications.

The thesis is roughly divided into two main sections. The first section involves updating the front page of the Reppi.fi service and simultaneously changing it to utilize client-side rendering. The second section examines the technologies used in the project and delves into the measurement of website statistics. Finally, the results of core web vital measurements between the old and new front pages are discussed.

Keywords: Angular, SPA, MPA, Single-page application, Multi-page application, Serverless, Server-side rendering

ALKUSANAT

Tämä opinnäytetyö on toteutettu yhteistyössä Reppi Oy:n kanssa, ja se käsittelee Server Side Renderingiä (SSR) Serverless ympäristössä sekä Reppi.fi palvelun uudistusta. Haluan esittää suuret kiitokset Reppi Oy:lle, että he mahdollistivat tämän opinnäytetyön toteuttamisen ja tarjosivat minulle arvokkaan tilaisuuden työskennellä heidän projektinsa parissa.

SSR tarjoaa tehokkaan tavan parantaa verkkosivustojen suorituskykyä ja hakukonenäkyvyyttä siirtämällä sivuston HTML:n (Hyper Text Markup Language) generoinnin takaisin backendille, vähentäen kuormaa käyttäjällä ja nopeuttamalla ensimmäisen piirron (engl. First Contentful Paint tai FCP) nopeutta.

Etusivu on usein verkkosivun tärkein osa, ja sen toiminnallisuudella on suuri vaikutus sivuston käyttäjäkokemukseen ja siihen palaako käyttäjät sivustolle. Opinnäytetyön toiminnallisen osuuden tavoitteena on tarjota Repille käyttökelpoinen ja toimiva ratkaisu, joka parantaa heidän verkkosivujensa suorituskykyä, käyttäjäkokemusta sekä hakukoneoptimointia. Työn aikana tutkittiin Angular-sovellusten rakennetta, SSR-mallin toteutusta Serverless tekniikalla sekä verkkosivujen statistiikkaa. Lisäksi keskityttiin uuden etusivun luomiseen, jotta se vastaisi paremmin Repin tarpeita ja brändiä.

Tässä opinnäytetyössä esitellään ensin projektin vaiheet, jonka jälkeen käydään läpi taustatietoa käytetyistä teknologioista, SPA- (Single Page Application) ja SSR-sovelluksista. Lopuksi “Verkkosivun statistiikka” luvussa arvioidaan toteutettujen muutosten vaikutuksia verkkosivun suorituskykyyn, hakukoneoptimointiin sekä käyttäjäkokemukseen.

Haluan kiittää kaikkia niitä, jotka ovat olleet mukana tämän opinnäytetyön tekemisessä, erityisesti Reppi Oy:tä.

SISÄLLYS

SYMBOLI- JA LYHENNELUETTELO

- SSR: Server-Side Rendering (Palvelinpuolen renderointi)
- SPA: Single Page Application (Yksisivu sovellus)
- SEO: Search Engine Optimization (Hakukoneoptimointi)
- API: Application Programming Interface (Sovellusohjelmointirajapinta)
- FCP: First Contentful Paint (Ensimmäinen visuaalinen sisältö)
- LCP: Largest Contentful Paint (Suurin visuaalinen sisältö)
- TBT: Total Blocking Time (Kokonaisaika, jolloin pääsäie on estynyt)
- HTTP: HyperText Transfer Protocol on verkkoprotokolla, joka mahdollistaa tiedon siirron ja kommunikoinnin tietokoneiden välillä, erityisesti selaimen ja verkkopalvelimen välillä, jotta voidaan pyytää ja näyttää verkkosivuja ja muita verkkoresursseja.
- i18n: lyhenne kansainvälisestä kääntämisestä tai "internationalization" englanniksi. Se viittaa prosessiin, jossa ohjelmisto, verkkosivusto tai muu sovellus suunnitellaan tai mukautetaan toimimaan eri kielillä ja kulttuureissa.
- JavaScript: Pääasiassa verkkoympäristössä käytettävä ohjelmointikieli
- HTML: Hyper Text Markup Language
- DOM: (Document Object Model) on web-kehityksessä käytetty ohjelmointirajapinta, joka mahdollistaa HTML-dokumentin rakenteen esittämisen hierarkkisena puuna, jolloin se voidaan dynaamisesti manipuloida ja muokata JavaScriptin avulla.
- Auth-guard: on ohjelmistokehityskäsite, joka liittyy tavallisesti käyttäjän tunnistamiseen ja oikeuksien valvontaan tietyn toiminnon tai resurssin suhteen.
- CMS: verkkosivustojen ja digitaalisten sisältöjen hallintajärjestelmä. Se mahdollistaa sisällön luomisen, muokkaamisen ja julkaisemisen verkkosivustoille ilman syvää teknistä osaamista.
- CDN: Content Delivery Network on verkkopalvelu, joka optimoi verkkosivustojen ja sovellusten suorituskyvyn jakamalla sisältöä useille palvelimille ja sijainneille ympäri maailmaa. CDN auttaa nopeuttamaan tiedostojen latausaikoja ja parantamaan käyttäjäkokemusta.

- Observable: Observable on käsite, joka liittyy reaktiiviseen ohjelmointiin. Se on tietovirta, joka voi tuottaa ja lähettää tietoa ajan kuluessa.
- Backend: Viittaa sovelluksen taustapuoleen, joka käsittelee tietokantaan tallennettua tietoa ja palvelimella tapahtuvaa toimintaa ja palauttamista sen frontend (client)-sovellukselle.
- Client: Tarkoittaa käyttäjän käyttämää sovellusta tai verkkosivustoa, joka on vuorovaikutuksessa palvelimen (backend) kanssa.
- Refaktorointi: ohjelmoinnin käsite, joka tarkoittaa koodin uudelleenjärjestelyä ja parantamista siten, että se säilyttää saman toiminnallisuuden mutta on helpommin ylläpidettävää, luettavaa ja mahdollisesti tehokkaampaa.

1 JOHDANTO

“Reppi perustettiin vuonna 2018, kun perustajajäsenet alkoivat luoda alustaa, joka soveltuisi mahdollisimman monipuolisesti eri urheilulajien käyttöön.
Alun perin Crossfit-kilpailujen monipuolista kisa- ja lajivalikoimaa varten ohjelmoitu alusta vastaa lähes kaiken tyyppisten urheilulajien tarpeisiin.” (Reppi, 2021)

Toimeksiantajalla on tarve modernisoida reppi.fi-palvelua, parantaa sivuston käyttäjäkokemusta, hakukoneoptimointia sekä uudistaa sen etusivua. Tavoitteena on tehdä sivustosta käyttäjäystävällisempi ja helpottaa käyttäjiä löytämään relevanttia sisältöä, sekä tuoda koko sivuston sisältö nopeasti ja mahdollisimman sujuvasti käyttäjälle. Tämä tavoite saavutettiin konvertoimalla sovellus puhtaasta SPA (Single-Page Application) -sovelluksesta SPA/SSR-hybridi sovellukseksi. Sovelluksen kääntäminen puhtaasta SPA:sta SSR-hybridi sovellukseksi, stale-while-revalidate välimuistitus sekä tehokas CDN:n käyttö tarjoaa nopeampaa sivuston toimintaa tuomalla käyttäjälle valmiiksi renderöidyn etusivun lähes välittömästi, joka käy ilmi luvun “Verkkosivun statistiikka” mittaustuloksissa.
SSR-malli parantaa myös sivuston hakukonenäkyvyyttä toimittamalla valmiin HTML-dokumentin suoraan asiakassovellukselle (engl. client application mutta jatkossa selain), sillä kaikki hakukonebotit eivät välttämättä voi suorittaa JavaScriptiä (Google Developers, 2023e).

Projektin keskeisenä tavoitteena on hyödyntää Angular Universalin Server-Side Rendering (SSR) -ominaisuuksia sivuston suorituskyvyn parantamiseksi. SSR:n avulla pyrimme erityisesti optimoimaan ensimmäisen näytön sisällön latausaikaa (FCP, First Contentful Paint) sekä minimoimaan sivun visuaalista epävakautta (CLS, Cumulative Layout Shift). Näitä termejä selitetään myöhemmin tässä työssä yksityiskohtaisesti luvussa ”Verkkosivun statistiikka”.

Tärkeä osa projektia kohdistuu myös hakukoneoptimointi (engl. search engine optimization, SEO) -parannuksiin. Pyrimme parantamaan sivuston hakukonenäkyvyyttä ja optimoimaan sen indeksoitavuutta hakukoneissa. Tämä auttaa lisäämään sivuston löydettävyyttä ja houkuttelevuutta hakutulosten joukossa, mikä voi kasvattaa sivuston orgaanista liikennettä ja laajentaa sen käyttäjäkuntaa. (Google Developers, 2023d)

Hakukonenäkyvyyden parantamisen lisäksi projektin päämääränä on parantaa sivuston yleistä suorituskykyä ja käyttäjäkokemusta. Lisäksi uudessa etusivussa keskityttiin parantamaan käyttäjien navigointia kilpailuihin ilmoittautumisen ja tulosten seurannan osalta, jotta käyttäjät voivat löytää nämä toiminnot vaivattomasti.

On tärkeää huomioida myös sivuston responsiivisuus, eli kyky reagoida eri ruudunkokoihin ja ikkunakokoihin sekä muutoksiin, sillä merkittävä osa käyttäjistä käyttää sivustoa mobiililaitteilla. Tämän vuoksi meidän oli varmistettava, että mobiilikäyttäjillä on vähintäänkin yhtä hyvä käyttökokemus kuin työpöytäkäyttäjillä.

Erityisesti käyttäjäpalautetta on huomioitu ja toiveita tuotu esille. Esimerkiksi etusivulle luotiin innovatiivinen kisakalenteri, joka tarjoaa käyttäjille suoran pääsyn kilpailun tärkeisiin tietoihin, kuten ajankohtiin, ilmoittautumisten tilaan ja sulkeutumispäiviin. Lisäksi kalenterista käyttäjä pääsee suoraan toimintoihin kuten kilpailujen tulostauluille, ilmoittautumaan kilpailuihin sekä ilmoittamaan omat tuloksensa. Tämä muutos auttaa parantamaan sivuston käyttökokemusta merkittävästi tarjoamalla selkeät ja helpot navigointipolut käyttäjille. Samalla Repin etusivusta tulee informatiivisempi, käyttäjäystävällisempi ja modernimpi, mikä voi houkutella enemmän kävijöitä ja pitää nykyiset käyttäjät tyytyväisinä. Etusivun ylläpitäminen on myös suunniteltu helpoksi ja joustavaksi, jotta ei tarvita sovelluskehittäjää ylläpitämään etusivun informaatiota, vaan kaikki dynaaminen data hoidetaan Directus (Directus, n.d.) CMS:n (Content Management System) kautta.

Työn merkittävät saavutukset olivat seuraavat:

- Server-Side Rendering: SSR on merkittävästi parantanut sivuston suorituskykyä ja käyttäjäkokemusta. Tämä muutos on mahdollistanut sivuston sisällön nopean latautumisen ilman pitkiä odotusaikoja.

- Parempi hakukonenäkyvyys: Koska sivuista renderöidään HTML-palvelimella ja tarjoillaan selaimelle valmiina tai osittain valmiina, sivuston hakukonenäkyvyys parani.

- Etusivun visuaalisen epävakauden täydellinen stabilointi. SSR:n myötä uuden etusivun CLS (Content Layout Shift) on käytännössä mittaustuloksissa nolla.

- Selkeämpi etusivu: Uusi etusivu tarjoaa käyttäjille selkeämmin esillä olevaa relevanttia tietoa. Tärkeitä navigointipolkuja on korostettu ja uusia on luotu, mikä tekee etusivun käytöstä miellyttävämpää, kun haluttuihin osioihin pääsee yhdellä selkeällä siirtymällä.

1.1 Tekoälyn käyttö tässä opinnäytetyössä

Tässä työssä on hyödynnetty ChatGPT tekoälyä parempien käännösten luomiseen, alustavan sisällysluettelon luomiseen, oikeinkirjoittamisen tarkastamiseen sekä oman tekstin muuttamista luettavampaan formaattiin mutta kuitenkin niin että sisältö on kuten olen sen itse tarkoittanut.

Olen huolehtinut sisällön alkuperän tarkastamisen sekä tekijänoikeuksien kunnioittamisesta. Kaikki lähteet ovat minun etsimiäni eikä tekoälyn käyttämiä lähteitä. Sisältö on lähtökohtaisesti omaa tuotostani, vaikka tekoäly olisi sitä muokannut.

2 PROJEKTIN VAIHEET

2.1 Projektin aloitus

Nykyinen sovellus toimii puhtaana SPA-sovelluksena, mutta tahtotila on siirtyä SSR-malliin. SSR-mallissa sivut saadaan valmiiksi renderöitynä käyttäjille, jolla käytännössä poistettaisiin sivuston visuaalinen epävakaus. Edellä mainittu muutos myös nopeuttaisi ensimmäistä piirtoa ja sen myötä saattaa parantaa käyttäjäkokemusta. Projektin alkuvaiheessa tutustuin nykyiseen Serverless toimintamalliin ja pohdimme ratkaisuja SSR renderöintiin. Päädyimme niin sanottuun hybridi SPA/SSR sovellukseen, jossa sivun HTML-dokumentti renderöidään palvelimella ja toimitetaan käyttäjän selaimeen, jossa Angular SPA-sovellus ottaa sen haltuun. Angular asettaa toiminnallisuuden sivulle sekä tarvittaessa renderöi sivun loppuun. Edellä mainittu prosessi kuvataan tarkemmin luvussa ”Vertailu SPA- ja SSR-tekniikoiden välillä”.

Backendin suunnittelussa piti ottaa huomioon useita seikkoja, kuten renderöinnin kesto ja sen vaikutus lambda-funktion toimintaan. Lambda-funktioiden toimintaa kuvataan myöhemmin luvussa ”Käytetyt teknologiat”. Välimuistin käyttöä varten piti suunnitella kuinka kauan haluamme pitää asioita välimuistissa ja tarjoilla sisältöä sieltä. Backendin suunnittelussa piti myös huomioida mitä kaikkea halutaan renderöidä backendillä ja mitä käyttäjän selaimessa.

Projekti alkoi Angular-version päivityksellä, mutta päivitysprosessissa tuli vastaan haasteita, erityisesti prosessi rikkoi muutaman kriittisen komponentin, jolla muodostettiin kilpailupareja WOD-liigan kilpailuihin, tässä käytettiin tekoälyavusteista Algolia (Algolia, n.d.) hakua, joka ei kuitenkaan ollut yhteensopiva uusimman Angular version kanssa. Tämän seurauksena joidenkin ominaisuuksien oli väliaikaisesti poistuttava käytöstä, ja lähdettiin etsimään korvaavia ratkaisuja näille toiminnoille. Löysimme uuden vastaavan toteutuksen, joka selkeytti nykyistä koodia ja antoi meille saman toiminallisuuden edelliseen verrattuna.

SSR:ää varten piti pystyttää oma Lambda-funktio, joka pyörittäisi Express-serveriä, joka taas renderöisi pyydetyn sivuston HTML:n. Selitän Lambdan toimintaa myöhemmin luvussa ”Käytetyt teknologiat” myöhemmin sekä edellä mainittua prosessia luvussa ”Vertailua SPA- ja SSR-tekniikoiden välillä”.

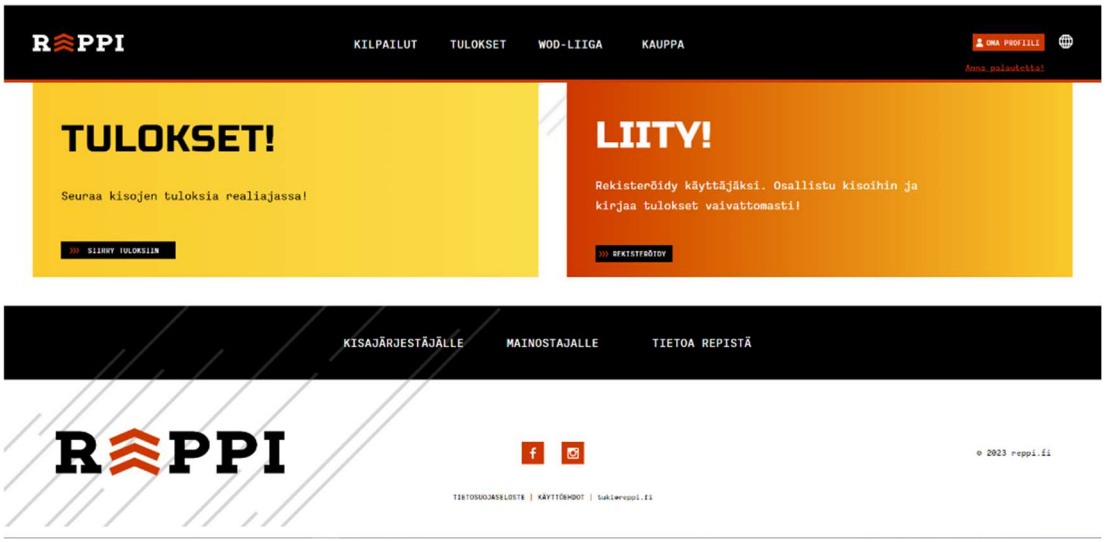
2.2 Eteneminen

Projektin aikana käytiin viikoittaisia statuspalavereja, jossa keskusteltiin mahdollisista käyttöliittymä (engl. User Interface, UI) muutoksista, backendin toiminnallisuuksien pystyttämisen etenemisestä sekä projektin tilasta. Käyttäjäpalaute oli myös projektin keskiössä ja statuspalavereissa tarkasteltiin saatua palautetta sivun ulkonäöstä ja toiminnasta. Saadun palautteen perusteella muokattiin sivuston toimintaa käyttäjäystävällisemmäksi. Koin statuspalaverit elintärkeäksi osaksi projektia ja ne auttoivatkin prosessissa huomattavasti. Viikoittainen palaute sivun ulkoasusta, rakenteesta sekä toiminnasta auttoi muokkaamaan koodia aikaisessa vaiheessa ja säästyimme suurilta refaktoroinneilta. Projektin alussa pystytimme sivulle oman ympäristön, jossa pystyimme testaamaan sivua käytännössä eri laitteilla ja käyttäjillä rinnakkain vanhan sivuston kanssa. Tämä mahdollisti nopean iteroinnin, testauksen ja palautteen saamisen testaajilta, sekä konkreettisen vertailun uuden ja vanhan sivuston välillä.

2.3 Edellisen etusivun haasteet ja niiden ratkaisut

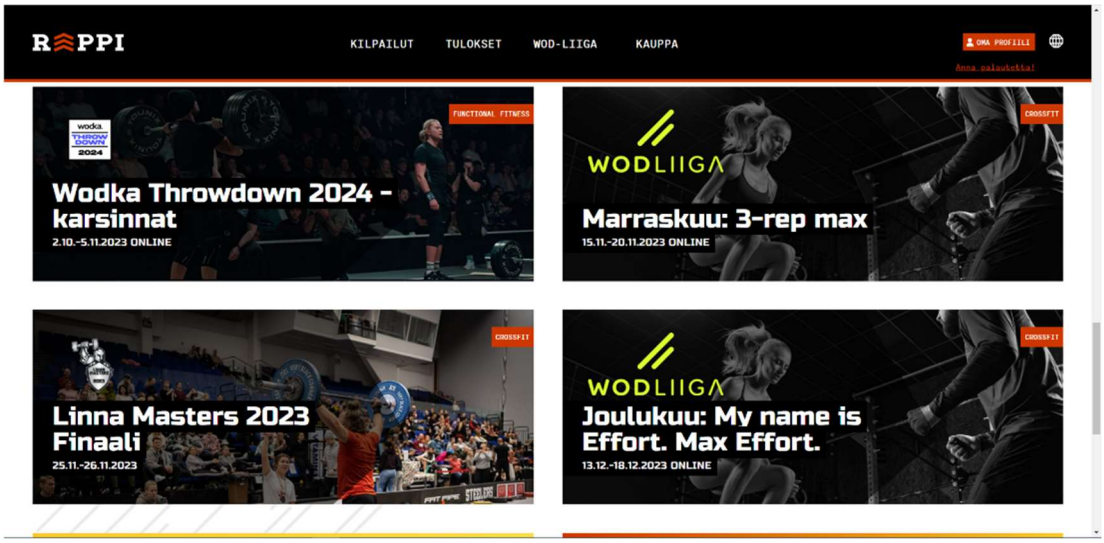
Yksi Repin uuden etusivun vaatimus oli mahdollisimman pieni sisällön visuaalinen epävakaus eli Cumulative Layout Shift (CLS), joka oli muodostunut haasteeksi edellisellä etusivulla sille saavuttaessa. Staattinen sisältö latautui ensin, jonka jälkeen usean sekunnin päästä loppusisältö oli latautunut ja puski koko sivua alaspäin aiheuttaen epämiellyttävän käyttäjäkokemuksen. Esimerkkinä rekisteröidy-nappi on suuressa laatikossa sivun alareunassa (kuva 1). Kilpailujen latauduttua rekisteröidy-nappi työntyy pois käyttäjän näkymästä (kuva 2), sillä kilpailut latautuvat napin edelle ja tästä johtuen käyttäjä saattaa navigoida vahingossa kilpailusivulle rekisteröitymissivun sijaan. Uudelle etusivulle

haluttiin myös valtava määrä dataa verrattuna edelliseen versioon, joten järkevä ratkaisu oli tässäkin tapauksessa siirtyä puhtaasta SPA-sovelluksesta SSR/SPA hybridi sovellukseen, jossa sivun HTML ensin tuotetaan backendillä, toimitetaan CloudFrontin (CDN) välimuistiin ja sen jälkeen toimitetaan valmiiksi luotu HTML-sivu käyttäjän selaimelle, ja lopuksi Angular asettaa siihen toiminnallisuuden (JavaScriptin) ja suorittaa selaimessa suoritettavat toimenpiteet.



Kuva 1 Reppi etusivun aloitustilanne

Huomaa suuri vierityspalkki oikealla kuvassa 2 ja sisällön määrä etusivulla. Alareunassa hivenen näkyvä rekisteröidy nappula on puskettu kaiken muun sisällön alapuolelle, tällaista visuaalista sisällön epävakautta kutsutaan layout shiftiksi (Google Developers, 2024b).



Kuva 2 Reppi etusivu datan latautumisen jälkeen

Kehitys myös puhtaasta SPA-sovelluksesta SPA/SSR-hybridisovellukseksi edesauttoi koko sivuston toimintaa, ei vain pelkästään etusivun. Samalla päivitettiin SPA-puolen toimintaa siihen suuntaan, että sovellus säilyttää selaimen muistissa tiedot aina kun mahdollista. Jos etusivulla haetaan kolme uusinta artikkelia niin artikkelien listaussivulla ei enää haeta näitä kolmea uusiksi vaan haetaan vain loput. Myöskään kaikkia kilpailuja ei haeta joka kerta uudestaan. Saavuttaessa kilpailujen listaussivulle edellisellä sivulla haetut tulokset säilytetään selaimen muistissa, vaikka navigoitaisiin sivuston muihin osiin. Sivuston käytettävyys ja käyttäjäkokemus on parantunut näin ollen huomattavasti, kun käyttäjä ei joudu odottamaan sisällön latautumista sovelluksin osissa missä hän on jo vieraillut aikaisemmin.

Myös hakukoneoptimointia haluttiin parantaa, joka tässä tapauksessa tapahtui suhteellisen orgaanisesti, kun siirrytään käyttämään SSR/SPA-hybridimallia. On myös hyvä huomioida, että vaikka käytän esimerkkinä Googlen aineistoa, niin hakukoneoptimointi paranee kaikissa hakukoneissa pelkästään jo sen takia että sivusto ei vaadi JavaScriptiä näyttääkseen informaatiota vaan kaikki tämä on jo valmiiksi renderöidyssä HTML-tiedostossa. Jotkin hakukoneet ovat alkaneet myös huomioida verkkosivujen keskeisiä laatuindikaattoreita (engl. Core Web Vitals) tuloksissaan, tarkoittaen että mitä paremmat mittaustulokset sivu saa sitä paremmin se sijoittuu vastaavaan sivuun verraten (Hines, 2021).

Näitä mittaustuloksia käydään läpi myöhemmin luvussa ”Verkkosivun statistiikka”.

2.4 Uuden etusivun toteutus

Tärkeimpänä ominaisuutena ja vaatimuksena uudelle etusivulle korostetaan kilpailukalenteria, jolla pyritään edistämään aktiivisten kilpailujen tunnettavuutta ja helpottamaan siirtymistä kilpailujen tuloksiin ja ilmoittautumisiin. Tavoitteena oli parantaa sivuston navigointia ja tarjota suorat linkit seuraaviin toimintoihin: kilpailun tulokset, kilpailuun ilmoittautuminen asianmukaisena aikana sekä mahdollisuus käyttäjän syöttää tuloksiaan suoraan kilpailuun. Tässä yhteydessä nousi esiin ongelma, joka johtui sivun rakenteesta: Käyttäjän pitää ilmoittaa kilpailujen tulokset lajikohtaisesti. Näin ollen käyttäjä olisi pitänyt navigoida ensin sopivaan kilpailuun ja sen jälkeen suodattaa tuloksia sen perusteella mikä tulos on vielä ilmoittamatta. Päädyttiin kompromissiin, jossa navigointi toteutettiin vain lajinäkymään, josta käyttäjä siirtyy itse lajiin, mihin haluaa ilmoittaa tuloksensa.

Kilpailukalenterin laajentaminen ja mukauttaminen osoittautui haastavaksi, sillä haluttiin mahdollistaa kilpailujen lisääminen kalenteriin, joita ei ollut järjestetty Repin kautta. Edellä mainitussa tapauksessa kisalla ei siis ole tietokannassa mitään tietoja kilpailusta, joten kilpailusta lisätään seuraavat tiedot suoraan kalenteriobjektiin Directuksessa: kilpailun nimi sekä alkamis- ja päättymispäivämäärä (muiden tietojen lisäksi). Jos kyseessä oli niin sanottu virallinen Repin kisa, nämä tiedot olisivat kalenteriobjektiin linkatussa kisassa tietokannan puolesta jo valmiiksi. Ei kuitenkaan haluttu, että näitä tietoja pitäisi syöttää käsin joka kerta kun luodaan kalenteriobjektia viralliselle kisalle. Ratkaisuksi valittiin selaimen ottavan nämä tiedot kalenteriobjektiin linkatusta kilpailusta, jos kyseinen objekti löytyy. Muussa tapauksessa tiedot lisätään suoraan kalenteriobjektiin Directuksen puolella.

Kilpailuihin ilmoittautumisessa esiintyi myös sivun rakenteellisia haasteita, koska tuki oli luotava kahdelle erityyppiselle ilmoittautumiselle: WOD-liigan

kilpailuille, joissa kilpailuun rekisteröidytään "kisapassin" avulla, ja muille kilpailuille, joissa voi olla osallistumismaksu tai ei. Näille kahdelle tyypille oli erilaiset navigointipolut. Ongelmaan löytyi onneksi yksinkertainen ratkaisu: tarkistettiin, oliko kilpailulla "liiga" -objekti. Jos tällainen objekti löytyi, kyseessä oli WOD-liigan kilpailu, ja navigoitiin WOD-liigan ilmoittautumispolkua pitkin. Muussa tapauksessa navigoitiin ns. normaalin kisan ilmoittautumisreittiä pitkin. Käyttäjän kirjautumisen tarkistamista painikkeen näyttämisen yhteydessä ei koettu tarpeelliseksi, sillä käyttäjäohjaus kulki Angulariin luodun auth-guardin kautta. Auth-guardin tehtävä on varmistaa, että käyttäjä olisi kirjautunut ja jatkaa navigointia sen jälkeen, jos käyttäjällä ei ole voimassa olevaa kirjautumistietoa käyttäjä ohjataan kirjautumissivulle. Auth-guardia vaativat sivut piti suorittaa täysin SPA-sovelluksena, sillä luonnollisesti emme halunneet kuljettaa kirjautuneen käyttäjän tietoja palvelimelle ja sieltä uudestaan selaimelle pelkästään sivuston HTML:ää varten.

Bannerikomponentin diaesityksen luominen tuotti lieviä haasteita, sillä haluttiin välttää tarvetta tehdä säännöllisiä palvelinpyyntöjä sekä toimintoja tietyin aikavälein. Sen sijaan haluttiin vaihtaa diaa selaimessa ilman ylimääräisiä palvelinkutsuja. Tämän ongelman ratkaisemiseksi käytettiin @Inject(platform-browser) -ominaisuutta, jolla tarkistettiin, toimiiko sovellus käyttäjän selaimessa vai palvelimella. Jos kyseessä oli palvelin, ei tehty mitään. Sen sijaan, jos kyseessä oli käyttäjän selain, käynnistettiin ajastin, joka vaihtoi diaa määritetyn aikavälin välein. Ajastinta piti kuitenkin pyörittää Angularin “ulkopuolella” sillä Angular ei koe olevansa vakaa, jos se suorittaa säännöllisiä toistuvia toimenpiteitä (Angular, n.d.-a), esimerkiksi diaesityksen dian vaihtoja. Tämä saavutettiin helposti käyttämällä NgZone.runOutsideAngular-palvelua suorittaakseen annetun funktion Angular-sovelluksen pääsilmukan “ulkopuolella”.

Angular-sovellukset toimivat yleensä yksisäikeisesti ja käyttävät pääsilmukkaa tiettyjen toimintojen, kuten muutosten seurannan ja näkymien päivittämisen, suorittamiseen. Tämä tarkoittaa, että kaikki sovelluksen koodi suoritetaan tässä pääsilmukassa, joten jos jokin pitkäkestoinen tai toistuva tehtävä suoritetaan siellä, se voi aiheuttaa sovelluksen hidastumista tai kaatumista. (Tagger, 2023)

Bannerin haluttiin olevan helposti ylläpidettävissä, mutta samalla monipuolinen. Päädyttiin tukemaan kolmea erilaista diaa:

- Sivun sisäinen mainosbanneri: Tämä tarkoittaa mainosta, joka on suoraan liittyvä sivuston sisältöön. Tämän tyyppinen dia voi sisältää esimerkiksi ajankohtaisia uutisia tai tarjouksia.

- Ulkoinen mainosbanneri: Tämä tarkoittaa mainosta, joka voi olla linkki ulkopuoliselle mainostajalle tai kumppanille. Tämän tyyppinen dia voi tuoda lisätuloja sivuston ylläpitäjille.

- Kilpailunoston dia: Tämä dia voi sisältää tietoa tulevista kilpailuista tai tapahtumista. Se tarjoaa mahdollisuuden korostaa tärkeitä kilpailuja ja houkutella osallistujia.

Tällainen monimuotoisuus antaa Repin ylläpitäjille mahdollisuuden tarjota erilaisia viestejä ja mainoksia käyttäjilleen, samaan aikaan tarjoten käyttäjille vaihtelevan ja mielenkiintoisen käyttökokemuksen.

Angularin platform-browser-moduuli tarjoaa tavan käsitellä selainkohtaisia asioita Angular-sovelluksessa. Tämä moduuli tarjoaa joukon työkaluja ja palveluita, jotka ovat hyödyllisiä selainympäristössä toimivissa sovelluksissa. Yksi tärkeä ominaisuus on PLATFORM_ID-tunnus (engl. Token), jota voi käyttää tarkistamaan, onko sovellus tällä hetkellä selaimessa vai muussa ympäristössä, kuten palvelimella.

@Inject(platform-browser) on Angularissa käytetty dekoraattori, joka mahdollistaa PLATFORM_ID-tunnuksen injektoimisen komponenttiin, palveluun tai muuhun Angular-sovelluksen osaan. Tämä tarkoittaa, että voit käyttää PLATFORM_ID-tunnusta kyseisessä komponentissa tai palvelussa.

Tässä (Ohjelma 1) on esimerkki siitä, miten @Inject(platform-browser) -dekoraattoria voidaan käyttää :

```
import { Inject, PLATFORM_ID } from '@angular/core';


constructor(@Inject(PLATFORM_ID) private platformId: Ob-
ject) {

  // Tarkistetaan, onko sovellus selaimessa

  if (isPlatformBrowser(this.platformId)) {

    // Suoritetaan selainkohtaiset toimenpiteet tässä

  }

}
```

Ohjelma 1. Platform-browser dekoraattorin käyttö

Tämä koodi injektoi PLATFORM_ID-tunnuksen ja käyttää isPlatformBrowser-funktiota tarkistamaan, onko sovellusalusta selain. Tämä mahdollistaa selainkohtaisten toimenpiteiden suorittamisen vain kun sovellus toimii selainympäristössä. Tämä voi olla hyödyllistä esimerkiksi silloin, kun halutaan välttää turhia HTTP-kutsuja palvelinpuolella tai suorittaa selainkohtaisia toimenpiteitä. Palvelimella tapahtuvaa renderöintiä haluttiin optimoida ja osaa renderöinnin kuormasta siirtää käyttäjän selaimelle. Vaikka palvelimella tapahtuva renderöinti on suhteellisen nopeaa, halusimme silti välttää suurta tiedostokokoa ja optimoida prosessia. Jatkokehitimme myös olemassa olevaa lambda-funktiota, joka optimoi kuvien kokoa. Kuvat muutetaan jatkossa WebP-formaattiin, joka säilyttää parhaan mahdollisen kuvanlaadun mutta minimoisi kuvien tiedostokokoa (Google Developers, 2023c). Tällä toiminnallisuudella muun muassa etusivun koko putosi noin kahdeksasta vain kahteen megatavuun.

Toisena esimerkkinä mainittakoon, että etusivu on erityisen pitkä pystysuunnassa ja koimme, että ei ole välttämätöntä renderöidä koko sivua palvelinpuolella, sillä käyttäjä ei todennäköisesti ehdi nähdä kaikkea sisältöä välittömästi vaan voimme suorittaa osan renderöinnistä käyttäjän selaimessa. Selain on todennäköisesti ehtinyt renderöidä näkymän käyttäjän selatessa sivua alaspäin.

Päätimme toteuttaa tämän ottamalla käyttöön seuraavan lähestymistavan: ensimmäiset kaksi komponenttia, diasesitys ja kilpailukalenteri, renderöidään palvelinpuolella, kun taas loput osat sivusta renderöidään selaimessa. Tätä varten otimme käyttöön Angulariin direktiivin (engl. directive) "app-shell-no-render". Käytännössä tämä direktiivi määrittää, renderöidäänkö komponentti palvelimella vai selaimessa.
Tämä lähestymistapa mahdollistaa sivun tehokkaamman latautumisen ja suorituskyvyn parantamisen, kun tarpeetonta palvelinpuolen renderöinti vältetään. Käytettävät resurssit voidaan kohdistaa niin, että ne käytetään vain ensimmäiseen nähtävään sisältöön.

Seuraavassa koodissa (Ohjelma 2.) on “App-shell-no-render" direktiivi kokonaisuudessaan

```
import { isPlatformServer } from '@angular/common';
import { Directive, Inject, OnInit, PLATFORM_ID, Tem-
plateRef, ViewContainerRef } from '@angular/core';

@Directive({
  selector: '[appShellNoRender]'
})
export class AppShellNoRenderDirective implements OnInit
{
  constructor(private viewContainer: ViewContainerRef,
private templateRef: TemplateRef<any>, @Inject(PLAT-
FORM_ID) private platformId) {}

  ngOnInit() {
    if (isPlatformServer(this.platformId)) {
      this.viewContainer.clear();
    } else {
      this.viewContainer.createEmbeddedView(this.tem-
plateRef);
    }
  }
}
```

Ohjelma 2. Direktiivin koodi

Selvyyden vuoksi, kuvassa 3 "rpi"-etuliitteellä alkavista komponenteista, jotka eivät ole saaneet "appShellNoRender" -direktiiviä, renderöidään serveripuolella. Tämä tarkoittaa käytännössä sitä, että käyttäjän ei tarvitse odottaa koko sivuston latautuvan serveripuolella ennen kuin hän voi alkaa käyttää sitä. Sen

sijaan sivusto toimittaa käyttäjälle ensin osan sisällöstä, ja käyttäjä voi selata sivustoa alaspäin samaan aikaan kun Angularin selainpuoli hoitaa loppujen osien renderöinnin. Tämä potentiaalisesti parantaa käyttökokemusta, optimoi HTML-tiedoston kokoa sekä lyhentää odotusaikoja serveripuolen renderöinnin suhteen.

```
<div class="front-page">
  <div class="front-page-content">
    <rpi-top-banner class="top-banner" [bannerSlides]="bannerSlides"></rpi-top-banner>
    <rpi-competition-calendar [usersCompetitionIds]="usersCompetitionIds" [competitions]="competitions" class="competition-calendar">
    </rpi-competition-calendar>
    <rpi-news *appShellNoRender class="news" [articles]="articles"></rpi-news>
    <rpi-competition-reports *appShellNoRender class="reports" [reports]="reports"></rpi-competition-reports>
    <rpi-logo-carousel *appShellNoRender class="logos" [logos]="logos"></rpi-logo-carousel>
    <rpi-viewer-tickets *appShellNoRender class="tickets"></rpi-viewer-tickets>
    <rpi-wod-league *appShellNoRender class="wod" [wodCompetitions]="wodCompetitions" [currentLeagueId]="currentLeagueId"></rpi-wod-league>
    <rpi-services-description *appShellNoRender class="services" [serviceDescriptions]="serviceDescriptions"></rpi-services-description>
    <rpi-reference *appShellNoRender class="reference" [references]="references"></rpi-reference>
  </div>
</div>
```

Kuva 3 Direktiivin käyttö

Valitettavasti sivuston nykyinen kielituen käännöstoiminto ei ollut yhteensopiva SSR:n kanssa. Emme saaneet mahdollisuutta ratkaista tätä ongelmaa asianmukaisesti ulkoisien aikataulupaineiden vuoksi. Ongelman ydin oli siinä, ettei ollut tehokasta tapaa kommunikoida samanaikaisesti sekä serverille että selaimelle siitä, kumpaa sivuston versiota (suomen- vai englanninkielistä) käytetään.

Ratkaisuna tähän ongelmaan päätimme lisätä sivuston URL-osoitteen loppuun /fi tai /en -polun, joka määräisi, kumpaa sivuston versiota käytetään selailuun. Tämä polku myös ilmoittaisi palvelimelle, kumpi sivuston versio on käyttäjälle toimitettava. Jos sivuston versio on suomenkielinen (/fi), polun loppuosa poistettaisiin käyttäjän näkyvistä URL-osoitteesta selkeyden vuoksi. Tämä päätös tehtiin, sillä käyttäjät todennäköisesti odottavat siirtyvänsä suomenkieliselle sivustolle, kun he saapuvat esimerkiksi osoitteeseen www.reppi.fi, eikä URL-osoitteiden näyttäminen muodossa www.reppi.fi/fi olisi ollut johdonmukaista.

Linkkien luomisen ja erityisesti jo olemassa olevien linkkien hallinnan (eli linkkien, jotka ovat luotu ja lähetetty ennen sivuston muuttamista hybridiksi) ja käsittelyn vuoksi meidän oli toistaiseksi ohitettava tämä ongelma. Tämä tarkoitti,

että kielipalvelu on väliaikaisesti poissa käytöstä, kunnes voisimme varata riittävästi aikaa ja resursseja ongelman pysyvään ratkaisuun.

3 KÄYTETYT TEKNOLOGIAT

3.1 Angular

Stackoverflow:n tekemän kyselyn mukaan Angular on yksi suosituimmista JavaScript-sovelluskehyksistä (Stackoverflow.com, n.d), joka on kehitetty helpottamaan yksisivusovellusten kehitystä. Angular on avoimen lähdekoodin sovelluskehys, jonka on kehittänyt Google. Angular on noussut suosioon sen monipuolisuuden ja käyttäjäystävällisyyden ansiosta.

Angularin tärkeimmät ominaisuudet ovat sen komponenttipohjainen rakenne ja reititysjärjestelmä, jotka mahdollistavat sovellusten kehittämisen modulaarisesti ja skaalautuvasti. Angular käyttää myös TypeScript-ohjelmointikieltä, joka mahdollistaa vahvasti tyypitetyn koodin kirjoittamisen, joka edesauttaa osaltaan virheiden löytämistä koodista ajoissa ja helpottaa sovelluksen ylläpitämistä. (Typescriptlang.org, 2023)

3.2 Directus

Directus on avoimen lähdekoodin tietokannan hallinta- ja sisällönhallintajärjestelmä (CMS), joka tarjoaa kehittäjille ja sisällöntuottajille työkaluja verkkosivustojen ja sovellusten sisällön hallintaan. Se eroaa monista perinteisistä CMS-järjestelmistä, sillä se antaa kehittäjille täyden hallinnan tietokantaan ja rajattoman joustavuuden sisällön rakenteen määrittelyssä. (Directus, n.d.)

Uudella etusivulla sekä koko sivuston laajuisesti on useita elementtejä, joita sisällöntuottajat voivat hallita. Etusivun komponenteista diaesitys, kisakalenteri, logokaruselli, palvelukuvaukset sekä referenssit hallinnoidaan

Directuksen kautta ja näiden ylläpitämiseen ei tarvita sovelluskehittäjää, vaan sisällöntuottajat itse voivat hallita sisältöä.

Directuksessa on RESTful JSON API, jonka avulla kehittäjät voivat liittää sen muihin sovelluksiin ja käyttää sisältöä millä tahansa alustalla.
Palvelu tarjoaa myös helppokäyttöisen käyttöliittymän, jonka avulla sisällöntuottajat voivat hallita sisältöä ilman syvällistä teknistä osaamista.

3.3 Serverless

Serverless on pilviteknologiaan liittyvä kehitys- ja käyttömalli, joka mahdollistaa sovellusten ja palvelujen suorittamisen ilman perinteisiä palvelimien konseptia, jolloin kehittäjät voivat keskittyä pelkästään koodin kirjoittamiseen ja sovellusten toiminnallisuuteen ilman huolta infrastruktuurista. Vaikka termi "serverless" voisi antaa sen kuvan, että servereitä ei ole lainkaan, se ei ole täysin totta. Serverless-arkkitehtuuri ei merkitse sitä, että palvelimia ei ole ollenkaan, vaan pikemminkin sitä, että kehittäjät eivät ole vastuussa palvelimien hallinnasta ja ylläpidosta. Termi "serverless" viittaa siihen, että infrastruktuuri ja palvelimet ovat piilossa ja abstrahoituja kehittäjiltä, jotka voivat keskittyä sovelluslogiikan kehittämiseen. (Red Hat. 2022)

Tärkeimmät ominaisuudet ja käsitteet serverless-mallissa ovat:

- Funktiot: Serverless-sovellukset koostuvat pienistä koodinpätkistä, joita kutsutaan usein funktioiksi. Nämä funktiot ovat itsenäisiä yksiköitä, jotka suorittavat tietyn tehtävän, kuten tiedonkäsittelyn, datan muokkauksen tai käyttäjäpyynnön käsittelyn. Esimerkki tunnetusta serverless-funktiopalvelusta on AWS (Amazon Web Services) Lambda.

- Maksu käytön mukaan: Serverless-mallissa kehittäjät maksavat vain käytetyistä resursseista ja suoritusajasta. Toisin sanoen, kehittäjää veloitetaan vain silloin, kun funktiot suoritetaan. Tämä voi tehdä

serverless-ratkaisuista taloudellisesti houkuttelevia, sillä heidän ei tarvitse maksaa turhasta kapasiteetista.

- Nopea käyttöönotto: Serverless-malli mahdollistaa nopean sovellusten kehityksen ja käyttöönoton, koska kehittäjät voivat keskittyä pelkästään koodiin ja sovellustoiminnallisuuteen, eikä heidän tarvitse huolehtia infrastruktuurin ylläpidosta.

- Vihreä pilvi: Kun ei luoda vain kehittäjän käyttöä varten fyysisiä servereitä hiilijalanjälki pienenee sillä resurssit ovat uudelleenkäytettäviä ja niitä kutsutaan vain käytön aikana. (Paul, 2023, s. 9)

Serverless-malli ei välttämättä sovi kaikkiin käyttötapoihin, ja sillä voi olla omat rajoituksensa, kuten suoritusaika- ja resurssirajoitukset. Esimerkkinä rajoituksista tässä projektissa jouduimme nostamaan Lambdan oletusarvoisia asetuksia muistista sekä suoritusajasta sillä ne eivät olleet riittäviä SSR-renderöintiä varten.

Tämä malli on hyvin suosittu erityisesti pienille ja nopeasti kasvaville sovelluksille, prototyypeille ja skenaarioille, joissa nopea kehitys ja alhaiset infrastruktuurikustannukset ovat tärkeitä. Monet suuret pilvipalveluntarjoajat, kuten Amazon Web Services, Microsoft Azure ja Google Cloud, tarjoavat serverless-palveluja.

3.4 Cloudfront

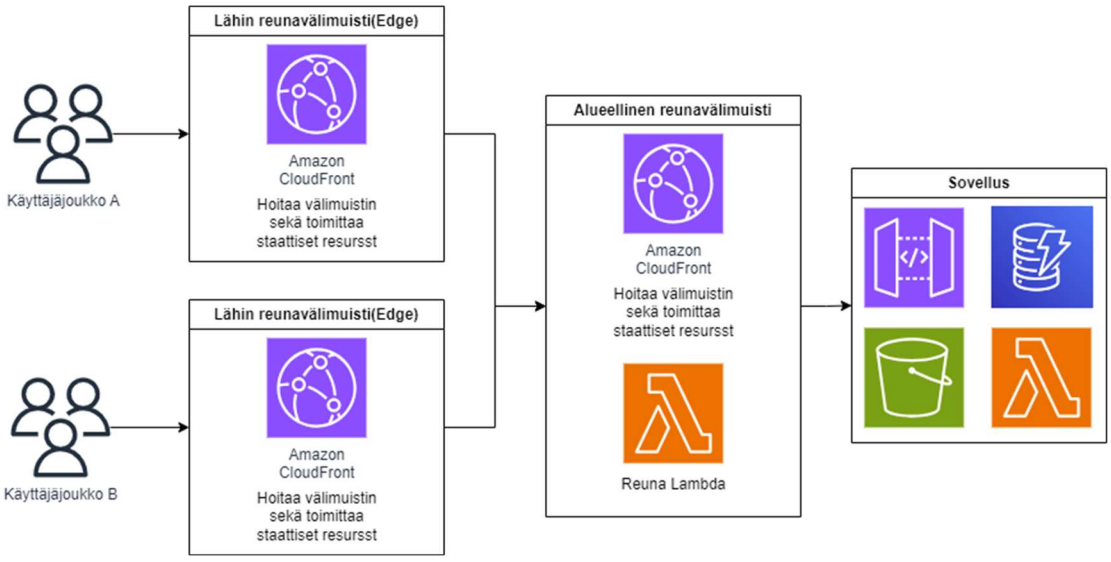
Amazon CloudFront(Amazon, n.d.-a) on CDN-palvelu, joka toimii maailmanlaajuisesti ja perustuu AWS:n verkostoon. Sen ydintoiminnot kohdistuvat staattisen sisällön jakeluun, kuten kuvien, CSS-tiedostojen ja JavaScriptin, jotta voidaan optimoida verkkosivustojen ja sovellusten suorituskykyä. CloudFront käyttää maailmanlaajuista Edge-asiakaskeskusverkkoa, joka vähentää latenssia ja nopeuttaa sisällön latausta eri puolilla maailmaa. Cloudfrontin toimintaa on havainnollistettu kuvassa 5.

Turvallisuuden näkökulmasta CloudFront tarjoaa tärkeitä suojatoimintoja, mukaan lukien DDoS-hyökkäyksien torjunnan ja SSL-salauksen. Nämä ovat ratkaisevassa roolissa verkkopalveluiden suojaamisessa ja käyttäjien tiedon turvallisuuden varmistamisessa. (Yutaka, 2021)

Dynaamisen sisällön jakelussa CloudFront on ratkaisu, joka nopeuttaa verkkosovelluksia ja API-palveluita. Tämä toimii yhteistyössä muiden AWS-palveluiden, kuten AWS Lambda ja Amazon S3, kanssa. Se mahdollistaa tehokkaan skaalautuvuuden ja joustavuuden eri käyttötapoihin.

Seurannan ja analytiikan osalta CloudFront tarjoaa tärkeitä työkaluja suorituskyvyn analysointiin ja käyttäjäkäyttäytymisen seuraamiseen. Seurantatiedot ovat yleensä avainasemassa palvelun optimoinnissa. Projektin aikana seurasimme tiiviisti CloudFrontin kuvavälimuistin kokoa, kun optimoimme kuvienkäsittely-lambdafunktion toimintaa. Yllätyimme positiivisesti, kuinka paljon pienemmäksi tämä välimuisti muuttui korkeankin liikenteen aikana.

Lisäksi CloudFront integroituu saumattomasti muiden AWS-palveluiden kanssa, mikä mahdollistaa hallinnan ja konfiguroinnin automatisoinnin. Tämä tekee CloudFrontista joustavan ja skaalautuvan ratkaisun erilaisiin käyttötapoihin. (Amazon, n.d.-b)



Kuva 4 Pelkistetty ylätason kaavio reunalokaatioista

Reunavälimuisti perustuu toimintaan, jossa resurssit tarjotaan käyttäjälle mahdollisimman läheltä heidän sijaintiaan. Esimerkiksi, jos tarkastellaan täysin kuvitteellista esimerkkiä, jossa käyttäjäjoukko A sijaitsee Pohjois-Euroopassa ja heitä palvelee Tukholmassa sijaitseva reunavälimuisti. Käyttäjäjoukko B voi myös olla Pohjois-Euroopassa, mutta he ovat lähempänä Helsingin reunavälimuistia. Molemmat joukot voivat käyttää samaa alueellista reunavälimuistia, esimerkiksi Pohjois-Euroopan alueellista reunavälimuistia. Tämä alueellinen reunavälimuisti ohjaa loppusovelluksen toimintaa ja voi esimerkiksi käynnistää Lambda-funktion kuvien kompressointia varten.

3.5 S3 bucket

Amazon S3 (Simple Storage Service) on keskeinen teknologia AWS-verkkopalveluiden tiedostojen ja datan tallentamiseen pilvessä.
Amazon S3 mahdollistaa joustavan ja skaalautuvan tiedon tallentamisen. Tämä palvelu on perusta monille verkkopalveluille ja sovelluksille, sillä se tarjoaa luotettavan tavan tallentaa ja hakea tiedostoja pilvestä. S3-palvelua voi käyttää monenlaisissa tarkoituksissa, kuten varmuuskopioinnissa, tiedostojen jakamisessa ja verkkosivustojen sisällön jakelussa.

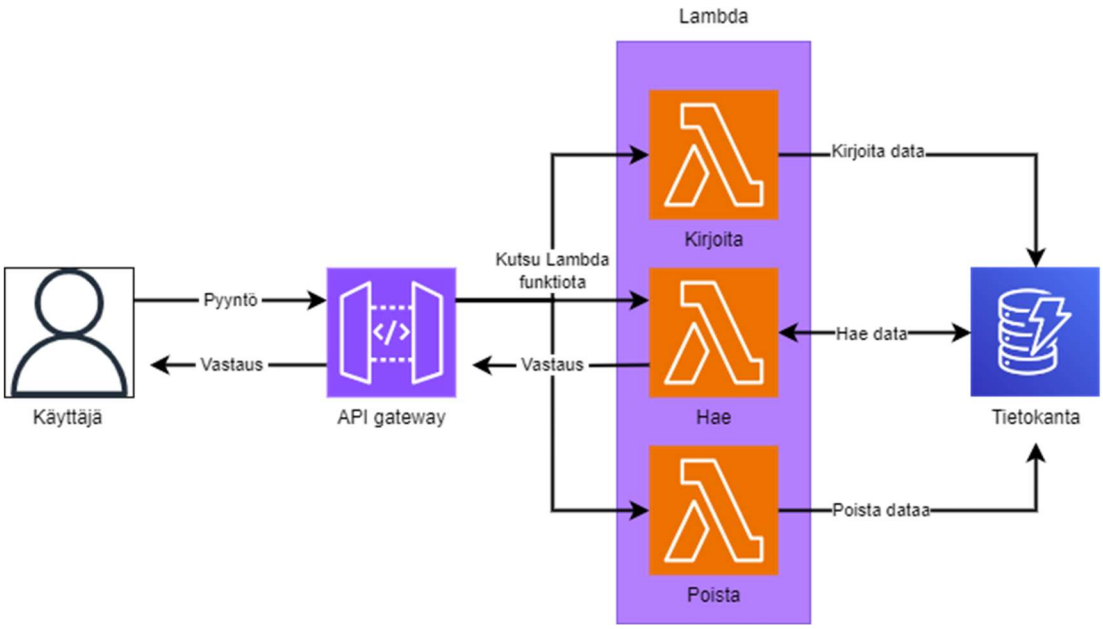
Käyttökohteina Amazon S3:lle voivat olla esimerkiksi verkkosivuston staattiset tiedostot, kuten HTML-tiedostot, kuvat, CSS-tiedostot sekä JavaScript kimput (engl. Bundlet). Tiedostot tallennetaan tietovarantoihin, joita kutsutaan nimellä "S3-buckets". Näiden tietovarantojen avulla voidaan järjestää ja hallinnoida sisältöä ja tietoa selkeästi ja tehokkaasti.

Amazon S3 tarjoaa myös korkeaa saatavuutta ja kestävyyttä. Tiedostot replikoidaan automaattisesti eri tietokeskuksiin ja alueille, mikä takaa, että tiedot ovat suojattuja ja saatavilla myös teknisten ongelmien sattuessa. (Amazon, n. d.-c)

3.6 Lambda

Amazon Lambda on merkittävä osa nykyaikaisten Serverless-sovellusten kehittämistä AWS-ympäristössä. Palvelu mahdollistaa toiminnallisuuden suorittamisen ilman tarvetta hallita perinteisiä palvelimia. Se toimii palvelumuotoisesti, mikä tarkoittaa, että ohjelmakoodi suoritetaan vasta, kun sille on tarvetta. Lambdaa voidaan käyttää laajasti erilaisissa sovelluksissa, kuten verkkosovellusten taustaprosesseissa, tietojen käsittelyssä ja automatisoiduissa tehtävissä. Esimerkkinä tästä kuvassa 6 havainnollistettuna lambdan toimintaa API Gatewayn ohjaamana.

Yksi AWS Lambda:n vahvuuksista on sen yhteensopivuus muiden AWS-palveluiden kanssa. Lambda voi toimia reaktiona erilaisiin tapahtumiin, kuten S3-tietovarannoissa tapahtuviin muutoksiin tai HTTP-pyyntöihin API Gatewayn kautta. Tämä mahdollistaa dynaamisten sovellusten ja palveluiden luomisen AWS-ekosysteemin sisällä.



Kuva 5 havainnollistettu Lambda toimintaa Serverless ympäristössä pelkistettynä ylätason kaaviona.

Lambda tarjoaa myös mahdollisuuden automatisoida tehtäviä ja prosesseja. Voit luoda ja aikatauluttaa Lambda-funktioita, jotka suorittavat tietyt tehtävät

automaattisesti. Esimerkiksi datan siirtäminen, raporttien luominen ja resurssien hallinta voidaan automatisoida Lambda-funktioiden avulla

Lisäksi AWS Lambda integroituu tiiviisti muihin AWS-palveluihin, kuten CloudWatch ja X-Ray, mikä mahdollistaa sovellusten seurannan, valvonnan ja tehokkuuden parantamisen. (Amazon, n.d.-d)

3.7 API Gateway

Amazon API Gateway mahdollistaa API-rajapintojen luomisen ja hallinnan pilvessä, mikä on keskeistä nykyaikaisissa sovelluksissa. Tämä palvelu tarjoaa monia hyötyjä, mukaan lukien rajapintojen hallinnan, valvonnan, turvallisuuden ja suorituskyvyn optimoinnin.

API Gateway mahdollistaa määrittelyn ja hallinnan erilaisten rajapintojen käyttötapauksiin, kuten RESTful- ja WebSocket-rajapintoja. Palvelulla voi myös suojata rajapintoja tunnistautumisen ja valtuutuksen avulla, mikä on tärkeää tietoturvassa.

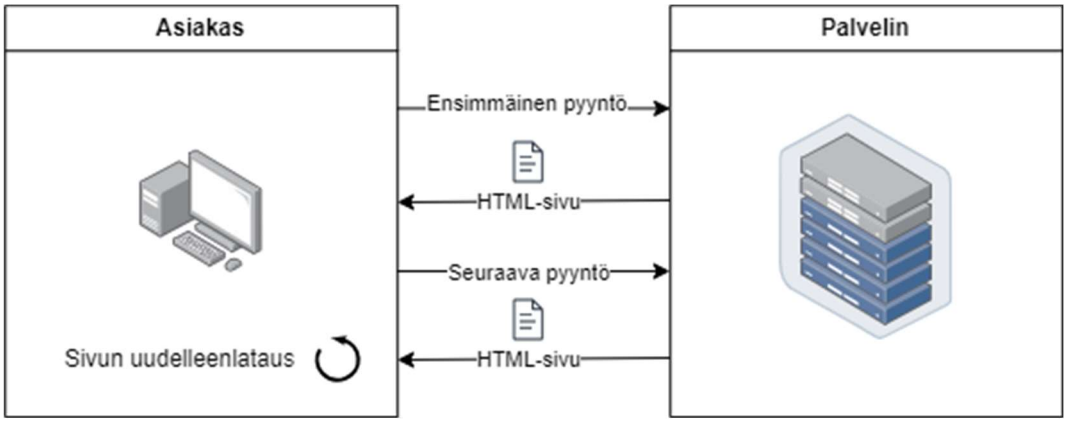
Lisäksi API Gateway tarjoaa laajan valikoiman työkaluja, kuten API-kehityskonsoleita ja integraatioita AWS-palveluihin, kuten AWS Lambda ja AWS S3, mikä helpottaa rajapintojen kehittämistä ja laajentamista. (Amazon, n.d.-e)

4 VERTAILU SPA- JA SSR-TEKNIIKOIDEN VÄLILLÄ

SPA (Single Page Application) ja SSR (Server-Side Rendering) versioiden välillä on tärkeää ymmärtää niiden eroja ja käyttötapoja. Kumpi niistä sopii parhaiten käyttäjälle, riippuu monista tekijöistä, kuten sovelluksen luonteesta, suorituskyvyn vaatimuksista ja kehitystiimin osaamisesta.

4.1 Verkkosivujen toiminnallisuuden historiaa

Verkkosivujen toiminnallisuuden historiaa voidaan jakaa karkeasti kahteen vaiheeseen: ennen ja jälkeen JavaScriptin yleistymisen. Ennen JavaScriptin yleistymistä perinteiset monisivuiset sovellukset (MPA, Multi page application) toimivat usein siten, että jokainen erillinen sivu ladataan erikseen käyttäjän selaimessa. Jokainen sivupyyntö vaati palvelimen lähettämään koko sivun uudelleen, jota on havainnollistettu kuvassa 7. Tämä oli raskasta ja suhteellisen hidasta sivujen latautumisen kannalta. Lisäksi tämä tarkoittaa, että kun käyttäjä navigoi sovelluksen eri osiin, sivut ladataan uudelleen joka kerta. Tämänlainen toiminta voi aiheuttaa käyttäjälle kokemuksen, että sivu “vilkkuu”.



Kuva 6 Perinteisen monisivuisen sovelluksen toiminnan kuvaaminen

Tämä voi aiheuttaa hieman hitaampaa käyttökokemusta ja hidastaa sovelluksen latausaikaa, sillä hyvin usein joudutaan lataamaan samaa sisältöä mikä olisi sivulla jo valmiiksi. Toisaalta monisivuiset sovellukset tarjoavat selkeän navigoinnin eri osiin ja niiden välillä. Ongelmana oli, että interaktiivisuus rajoittui usein sivujen väliseen navigointiin.

JavaScriptin yleistyminen ja selainten suorituskyvyn parantuminen mahdollistivat monimutkaisempien ja interaktiivisempien verkkosivujen kehittämisen. Tämä taas johti SPA-sovellusten suosioon (Davidson, 2023).

SPA:ssa perussivurakenne ladataan kerran, ja sen jälkeen sisältöä päivitetään dynaamisesti JavaScriptin avulla ilman tarvetta lähettää koko sivua uudelleen

palvelimelta. SPA-sovellukset voivat hakea ja päivittää tietoja palvelimelta ilman täydellistä sivun uudelleenlatautumista, mikä voi parantaa käyttäjäkokemusta ja nopeuttaa sovellusten toimintaa.

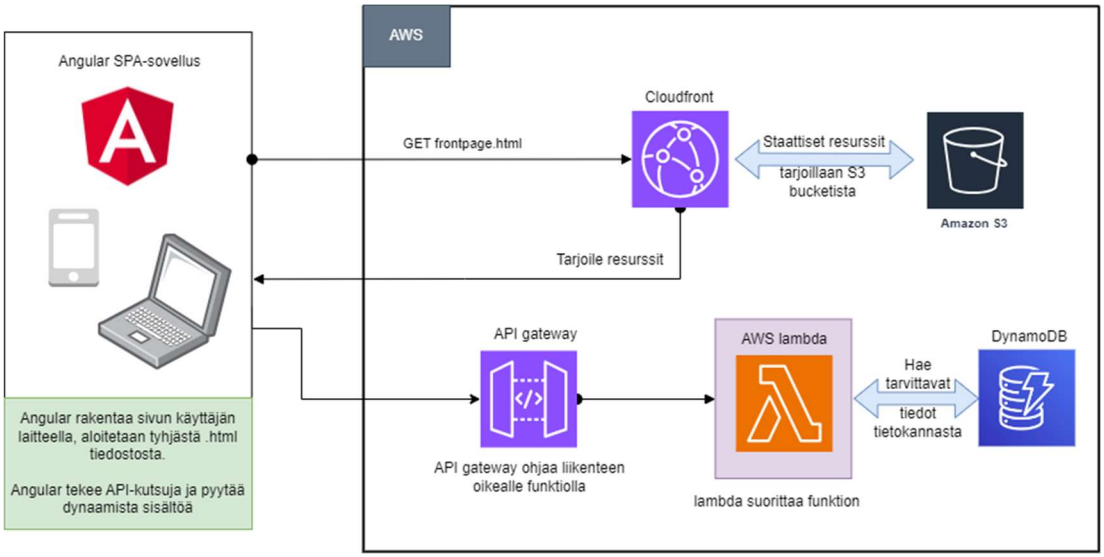
Yleensä SPA-sovelluksissa on yksi runkosivu, joka sisältää sovelluksen rungon, ja sen sisältöä päivitetään dynaamisesti käyttäjän toiminnan perusteella. Tämä antaa kehittäjille mahdollisuuden luoda monimutkaisempia ja dynaamisempia sovelluksia.

JavaScriptin ja SPA-sovellusten yleistyminen on merkinnyt merkittävää kehitystä verkkosivujen toiminnallisuuden historiassa. Tämä kehitys on mahdollistanut monipuolisempia ja käyttäjäystävällisempiä verkkosovelluksia ja tehnyt verkkokehityksestä entistä dynaamisempaa ja tehokkaampaa.

4.2 Single Page Application (SPA)

Yksisivuinen sovellus (Single Page Application, SPA) on verkkosovellustyyppi, joka toimii yhdellä HTML-sivulla, päivittäen dynaamisesti sisältöä uudelleenlataamatta koko sivua. SPA-sovellukset ladataan käyttäjän selaimessa kokonaisuudessaan. Data haetaan erinäisillä API-kutsuilla ja näytetään sivulla yleensä JavaScriptin avulla. Tämä tarkoittaa, että kaikki näkymät ja komponentit ladataan ja hallitaan selainpuolella. SPA:n tärkein etu on nopea ja saumaton käyttäjäkokemus, sillä siinä ladataan vain tarvittavat tiedot, tarjoten responsiivisen kokemuksen, jopa hitailla verkkoyhteyksillä. Lisäksi, koska SPA-sovellukset joutuvat uudelleenlataamaan vain tarvittavat tiedot, ne ovat tehokkaampia kuin perinteiset monisivuiset verkkosovellukset, joissa koko sivu ladataan uudelleen jokaisen muutoksen yhteydessä. (Freeman, 2022)

SPA:ssa sivun päivitykset eivät yleensä vaadi täyttä sivun uudelleenlatausta. Sen sijaan ne suoritetaan dynaamisesti JavaScriptillä, mikä yleensä tekee sovelluksesta nopeamman ja sulavamman tuntuisen. Kuvassa 8 on havainnollistettuna SPA-sovelluksen toimintaa Repin kontekstissa.



Kuva 7 Serverless SPA-sovelluksen toiminnan kuvaaminen

Yksisivusovellukset vaativat kuitenkin tarkkaa suunnittelua ja kehitystä. Kehittäjien täytyy varmistaa, että sovellus on helppo käyttää ja että käyttäjät voivat löytää kaikki tarvitsemansa toiminnallisuudet yhdeltä sivulta. Sovellus tulee myös optimoida suorituskyvyn kannalta, jotta latausajat pysyvät lyhyinä mobiilissakin. (Adobe Experience Cloud Team, 2023b)

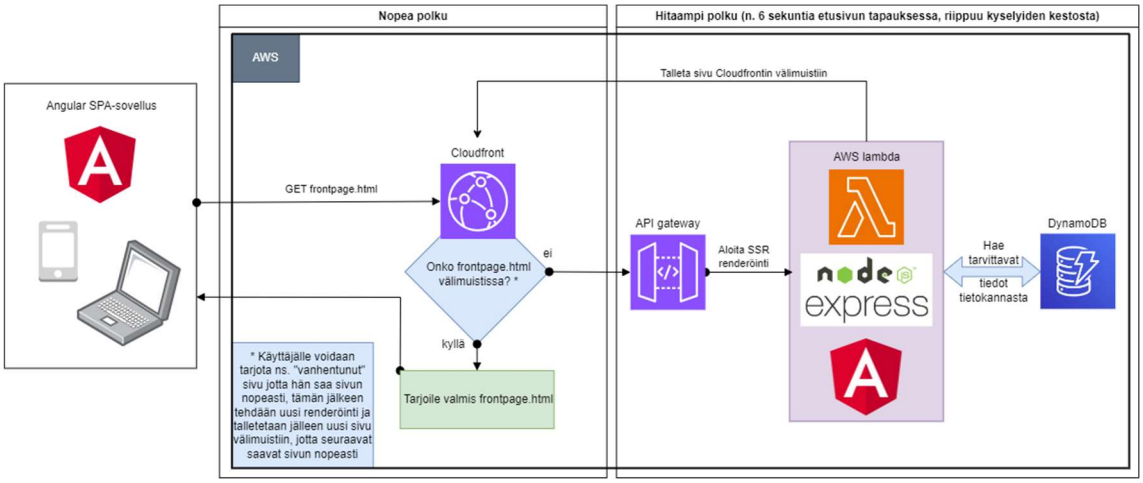
4.3 Server-Side Rendering (SSR)

SSR on lähestymistapa verkkosovellusten rakentamisessa, jossa sivujen HTML-sisältö generoidaan palvelimella ja lähetetään selaimelle valmiina HTML-tiedostona.

Tämä tarkoittaa, että selain saa sivun sisällön valmiina ja voi näyttää sen heti ilman, että tarvitsee odottaa JavaScriptin latautumista tai suorittamista. Tässä on tärkeää huomata, että vaikka sivu voi näyttää ja vaikuttaa toimivalta se ei kuitenkaan välittömästi sitä ole. Perusnavigointi voi olla toimivaa mutta JavaScriptiä vaativat toiminnallisuudet eivät ole heti valmiina. SSR-sovellukset tekevät sivujen renderöinnin palvelimella ennen kuin ne lähetetään selaimelle. Tämä tarkoittaa, että selain saa valmiin HTML-sivun suoraan palvelimelta. Hakukoneiden sivujen indeksointi on myös helpompaa ja suoraviivaisempaa SSR sovelluksissa, sillä JavaScriptiä ei tarvitse suorittaa informaation näyttämiseksi

ja tämän myötä SEO paranee. SSR-sovelluksilla voi olla myös parempi ensilatausaika verrattuna SPA-sovelluksiin, sillä käyttäjä saa melkein välittömästi sivun esirenderöidyn version. (Angular, n.d.-b)

Kun käyttäjä tekee HTTP-pyynnön sivusta, Cloudfront ensin tarkistaa onko kyseinen tiedosto jo välimuistissa. Jos tiedosto löytyy välimuistista, se palautetaan selaimelle suoraan. Tämän jälkeen tarkistetaan, onko sivu edelleen voimassa vai onko se vanhentunut, mikä riippuu sille asetetusta Stale-While-Revalidate(SWR) aikarajasta. Stale-While-Revalidate on HTTP-viestien välimuistin käytössä oleva ohje, joka mahdollistaa vanhan välimuistissa olevan tiedon palauttamisen samalla, kun uutta tietoa haetaan taustalla. SWR auttaa sivustojen nopeuttamisessa ja käyttäjäkokemuksen parantamisessa, sillä se mahdollistaa vanhan sisällön näyttämisen lähes välittömästi ja päivittää sen sitten taustalla uudella tiedolla (Posnick, 2019). Jos sivu on vanhentunut, API Gatewaytä kutsutaan, mikä puolestaan laukaisee lambda-funktion. Tässä vaiheessa Angularin Express-palvelin esirenderöi pyydetyn sivun ja toimittaa sen Cloudfrontin kautta selaimeen. Cloudfrontissa sisältö talletetaan välimuistiin, josta se voidaan tarjota seuraavalle käyttäjälle suoraan. Lopuksi kun sisältö on saapunut selaimelle, Angular lisää tarvittavan JavaScriptin sivulle toiminnallisuuden lisäämiseksi, sekä tekee muut tarvittavat muutokset mitä ei voida tai haluta suorittaa serverin puolella. Kuvassa 9 havainnollistettu edellä mainittua toimintaa eritoten Repin kontekstissa.



Kuva 8 Serverless SSR/SPA-sovelluksen toiminnan kuvaaminen

Käytännössä tämä sovellus on SPA/SSR-hybridi, joka yhdistää molempien sovellustyyppien parhaita puolia. Käyttäjän ei tarvitse odottaa pitkiä HTTP-pyyntöjä datan lataamiseksi, sillä ne ovat ideaalitilanteessa käsitelty jo valmiiksi palvelinpuolella, ja siten sisältö toimitetaan nopeasti. Tämä lähestymistapa parantaa myös sivuston hakukonenäkyvyyttä, sillä tavanomaisessa SPA-sovelluksen alkutilanteessa sivu on tyhjä, ja Angular rakentaa HTML:n ja JavaScriptin siihen. SSR:ssä sen sijaan on esirenderöity HTML-sivu, johon Angular lisää toiminnallisuuden ja puuttuvan sisällön, jos koko sivua ei renderöidä palvelimella. Tällaista toimintaa kutsutaan hydraatioksi, tarkalleen ottaen non-destructive hydraatioksi. Tarkoittaen seuraavaa: kun käyttäjä saa ensin palvelinpuolen generoiman sivun ja selaus käynnistyy, selaimen JavaScript voi ladata lisäsisältöä, tehdä pyyntöjä palvelimelle ja päivittää sivun dynaamisesti ilman, että koko sivua tarvitsee ladata tai rakentaa uudelleen. Tämä prosessi on "non-destructive hydration", sillä se ei tuhoa alkuperäistä sivun tilaa, vaan rikastaa sitä.

4.4 Vertailu

- Suorituskyky: SPA-sovellukset voivat olla nopeampia sivulla siirtyessä, sillä ne välttävät sivujen täydelliset uudelleenlataukset. SSR voi kuitenkin tarjota paremman ensilatausajan ja SEO-edut silloin kun käyttäjälle tarjoillaan koko HTML-sisältö kerralla eikä sitä rakenneta käyttäjäsovelluksessa. Tekniikat voidaan myös yhdistää hybridisovellukseksi, jossa molemmat hyvät puolet käyvät toteen.

- Kehitysnopeus: SPA-kehitys voi olla nopeampaa ja yksinkertaisempaa, sillä se perustuu pääasiassa JavaScriptiin ja API-palveluihin. (Adobe Experience Cloud Team, 2023a) SSR-kehitys vaatii enemmän huomiota palvelimen puolella ja voi olla monimutkaisempaa. Varsinkin tämän projektin puitteissa palvelinpuoli vaati huomattavaa kehitys- ja tutkimustyötä, esimerkiksi lambda-funktion ja sen asetusten asettaminen sekä pystyttäminen Express-palvelinta varten, jotta etusivu voidaan renderöidä SSR:n vaatimalla tavalla.

- SEO: Jos SEO on tärkeää, SSR on yleensä parempi vaihtoehto, sillä se tarjoaa valmiin HTML-sivun hakukoneille.

Lopullinen valinta riippuu projektin tarpeista ja resursseista. Monet yritykset käyttävät hybridiä molemmista lähestymistavoista tarpeen mukaan, jolloin esimerkiksi kriittiset sivut, joiden kävijämäärä on suurempi ja ovat relevantteja sivuston toiminnan ja näytettävän informaation kannalta, voivat olla SSR:ää ja vähemmän kriittiset ja pienemmän liikenteen osa-alueet voivat olla SPA:ta. Suunniteltaessa olisi tärkeää harkita käyttäjäkokemusta, suorituskykyä, kehitysaikaa sekä ylläpito- ja kehityskustannuksia.

SPA siis keskittyy sivujen dynaamiseen päivittämiseen JavaScriptin avulla, kun taas SSR generoi sivujen sisällön valmiiksi palvelimella ennen sen lähettämistä selaimelle. Kummallakin lähestymistavalla on omat vahvuutensa ja heikkoutensa, ja niiden valinta riippuu projektin vaatimuksista ja tavoitteista.

5 VERKKOSIVUN STATISTIIKKA

5.1 Cumulative Layout Shift

Cumulative layout shift (CLS) on verkkosivujen suorituskyvyn mittari, joka arvioi visuaalisten muutosten määrää ja sijaintia sivun latautuessa. CLS mittaa kuinka paljon sivun elementit siirtyvät tai muuttuvat odottamattomasti, kun käyttäjä on vuorovaikutuksessa sivuston kanssa, esimerkiksi kun sivua ladataan tai rullaillaan alaspäin. CLS on arvio siitä, kuinka häiritseviä nämä muutokset ovat käyttäjän kokemukselle.

5.1.1 Olennaisia asioita Cumulative Layout Shiftin mittaamisesta

Layout shift tarkoittaa sitä, kun verkkosivun elementit siirtyvät odottamattomasti paikasta toiseen, mikä voi aiheuttaa häiritsevän ja epämiellyttävän käyttäjäkokemuksen. Esimerkkejä ovat mainosten latautuminen, jotka muuttavat sivun rakennetta tai sisältöä, tai kuvien latautuminen, joka saa sisällön liikkumaan.

Kumulatiivinen vaikutus taas mittaa näiden layout shiftien yhteisvaikutusta sivun latauksen aikana. Pisteytys lasketaan sen mukaan, kuinka suuri muutos on ja kuinka lähellä käyttäjän näkökenttää se tapahtuu. Tämä antaa yhteisvaikutuksen, joka kuvastaa, kuinka paljon sivun ulkoasu muuttuu käytön aikana.

Mittayksikkönä Google mittaa CLS:ää pisteytyksenä. Sen mukaan hyväksyttävä arvo on yleensä alle 0,1. Suurempi arvo tarkoittaa suurempaa visuaalista sisällön epävakautta ja käytännössä yleensä huonompaa käyttäjäkokemusta. (Google Developers, 2024b)

5.1.2 Miksi cumulative layout shift on tärkeä mittari?

CLS on tärkeä mittari, sillä se liittyy suoraan käyttäjäkokemukseen. Jos sivuston elementit siirtyvät odottamattomasti ja aiheuttavat käyttäjälle sekaannusta, se voi johtaa käyttäjän turhautumiseen ja jopa siihen, että käyttäjä poistuu sivustolta vahingossa tai pahimmassa tapauksessa tarkoituksenmukaisesti.
Google ja mahdollisesti jo muut hakukoneet ovat alkaneet ottaa CLS:n huomioon hakutulosten sijoittelussa. Hyvä CLS-pisteytys voi auttaa parantamaan sivuston hakukonenäkyvyyttä. (Google Developers, 2023a)

Sivuston kehittäjien ja suunnittelijoiden tulisi seurata CLS:ää ja tehdä tarvittavia muutoksia sivuston rakenteeseen ja elementtien lataukseen, jotta voidaan minimoida häiritsevät layout shiftit ja parantaa käyttäjäkokemusta.

5.2 Search Engine Optimization

SEO eli suomeksi hakukoneoptimointi on prosessi, jonka avulla verkkosivuston tai verkkosisällön tavoitteena on parantaa näkyvyyttä hakukoneissa, kuten Google, Bing ja Yahoo. SEO:n tavoitteena on saada verkkosivusto näkymään korkeammalla tietyillä avainsanoilla ja aihepiireissä, mikä puolestaan lisää sivuston orgaanista liikennettä ja kävijämäärää. Tärkeitä SEO-asioita ovat:

- Avainsanat: SEO:ssa keskeinen osa on oikeiden avainsanojen valinta. Avainsanat ovat termejä tai lauseita, joita ihmiset käyttävät hakukoneissa etsiessään tietoa. Hyvä SEO-strategia keskittyy löytämään relevanteimmat ja kohdeyleisölle tärkeimmät avainsanat, joihin pyritään sijoittumaan hakutuloksissa.

- Sisällön optimointi: Verkkosivuston sisältö on keskeinen osa SEO:ta. Sisällön tulee olla laadukasta, informatiivista ja vastata käyttäjien kysymyksiin. Se tulisi myös optimoida avainsanojen mukaan, mutta luonnollisesti ja mielekkäästi.

- Tekninen SEO: Tämä osa keskittyy sivuston tekniseen rakenteeseen ja suorituskykyyn. Tekninen SEO sisältää asioita kuten sivuston nopeuden optimoinnin, mobiiliystävällisyyden, URL-rakenteen, ja hakukoneiden indeksointiongelmien korjaamisen. Useimmat sivustot läpäisevät teknisen SEO:n osuuden ilman mitään ylimääräisiä toimenpiteitä.

- Analytiikka ja seuranta: SEO-prosessia ei voi parantaa ilman tietoa siitä, mitä toimii ja mikä ei. Siksi olisi tärkeää seurata liikennettä ja tuloksia analytiikkatyökalujen, kuten esimerkiksi Google Analyticsin, avulla.

SEO on jatkuva prosessi, ja hakukoneet päivittävät säännöllisesti algoritmejaan, mikä voi vaikuttaa sivuston näkyvyyteen. Hyvä SEO voi auttaa sivustoa houkuttelemaan enemmän orgaanista liikennettä ja saavuttamaan liiketoiminnallisia tavoitteita. (Google Developers, 2023b)

5.3 First Contentful Paint ja Largest Contentful Paint

First Contentful Paint (FCP) ja Largest Contentful Paint (LCP) ovat kaksi tärkeää mittaria, jotka liittyvät verkkosivustojen suorituskyvyn arviointiin ja käyttäjäkokemukseen. Ne kuvaavat, kuinka nopeasti sivusto latautuu ja näyttää ensimmäisen ja suurimman näkyvän sisällön käyttäjälle.

5.3.1 First Contentful Paint

FCP mittaa aikaa, joka kuluu siitä hetkestä, kun käyttäjä aloittaa sivuston lataamisen, siihen hetkeen, kun selain ensimmäisen kerran piirtää sivulle jotain visuaalista sisältöä.

FCP antaa käyttäjälle ensivaikutelman sivuston nopeudesta. Mitä nopeammin FCP tapahtuu, sitä parempi käyttäjäkokemus on, sillä käyttäjä näkee jotain sivusta latautuvan välittömästi sen sijaan, että tuijottaisi tyhjää ruutua odottaessaan. (Google Developers, 2022b)

5.3.2 Largest Contentful Paint

LCP mittaa aikaa, joka kuluu suurimman näkyvän sisällön latautumiseen sivustolla. Suurin sisältö voi olla esimerkiksi iso kuva, video tai tekstielementti.

LCP on erityisen tärkeä mittari, sillä se antaa käsityksen siitä, kuinka nopeasti sivusto näyttää kaikkein merkittävimmän sisällön, joka kiinnostaa käyttäjää. Pitkät LCP-ajat voivat johtaa huonoon käyttäjäkokemukseen ja korkeisiin poistumisprosentteihin.

Sivuston suorituskyvyn optimointi pyrkii parantamaan FCP- ja LCP-arvoja, jotta sivuston elementit latautuvat nopeasti ja tarjoaa miellyttävän käyttäjäkokemuksen. Tämä voidaan saavuttaa esimerkiksi optimoimalla kuvia, käyttämällä tehokkaampia palvelimia, minimoimalla JavaScriptin vaikutukset, ja

varmistamalla, että suurin sisältö latautuu nopeasti. Esimerkiksi Repin sivustolla minimoimme kuvien tiedostokokoa siirtymällä Webp-formaattiin.

Google ja mahdollisesti muut hakukoneet ottavat myös huomioon FCP- ja LCP-arvot sivustojen arvioinnissaan ja sijoittelussa hakutuloksissa, joten ne vaikuttavat myös SEO:n (hakukoneoptimointiin). (Google Developers, 2023a)

5.4 Total Blocking Time

Total Blocking Time (TBT) on suorituskykymetriikka, jota käytetään arvioimaan verkkosivustojen latausnopeutta ja käyttäjäkokemusta. TBT mittaa aikaa, jonka aikana pääsäie (engl. main thread) on estynyt sivun latautumisen aikana. Pääsäie on se osa selainprosessista, joka käsittelee sivun renderöintiä, ja sen estyminen voi aiheuttaa hidastumista ja epämiellyttävän käyttäjäkokemuksen.

TBT:n tarkoituksena on tunnistaa ja mitata sivuston suorituskykyongelmia, jotka johtuvat JavaScriptin suorituksesta ja muista tehtävistä, jotka estävät pääsäiettä vastaamasta nopeasti käyttäjän vuorovaikutukseen. Tällaiset tehtävät voivat olla esimerkiksi:

- Pitkät suoritukset: Suuri määrä JavaScript-koodia tai monimutkaiset laskennalliset tehtävät voivat estää pääsäiettä ja hidastaa sivun latautumista.

- Pitävät tehtävät: Joissakin tapauksissa selain voi pitää tehtäviä odottamassa esimerkiksi resurssien latautumista ennen kuin se voi jatkaa sivun renderöintiä.

- Käyttäjän vuorovaikutuksen viive: Jos pääsäie estyy pitkäksi aikaa, se voi johtaa siihen, että käyttäjän vuorovaikutukseen vastaaminen, kuten napsautuksiin tai vieritykseen reagoiminen viivästyy, mikä huonontaa käyttäjäkokemusta.

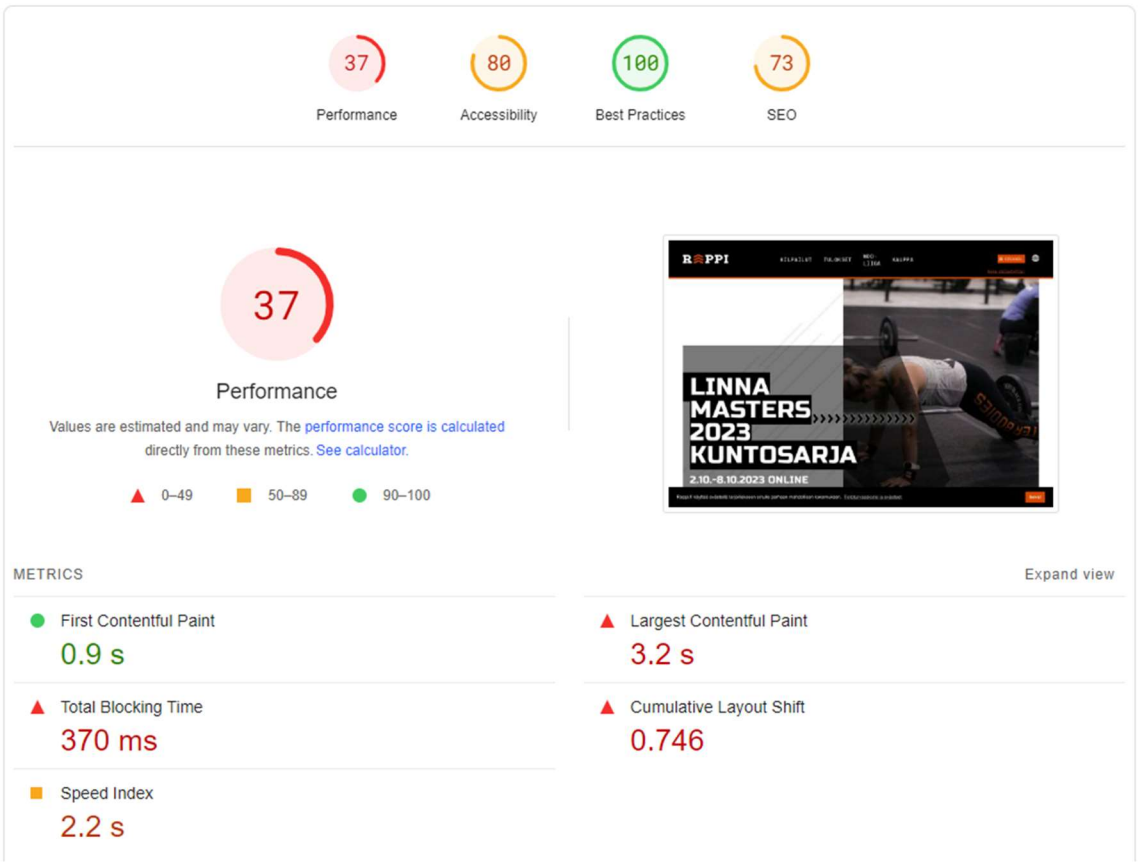
TBT:llä on suuri merkitys verkkosivustojen optimoinnissa ja suorituskyvyn parantamisessa. Pieni TBT-arvo on indikaattori siitä, että sivusto reagoi nopeasti käyttäjän toimiin, mikä voi parantaa käyttäjäkokemusta ja pitää sivuston käyttäjät tyytyväisinä. TBT:hen liittyvät parannukset voivat sisältää JavaScriptin optimoinnin, resurssien aikaansaamisen aikaisemmin latautumaan ja resurssien käytön vähentämisen kriittisissä renderöintitehtävissä. (Google Developers, 2022a)

5.5 Etusivun mittaustulokset

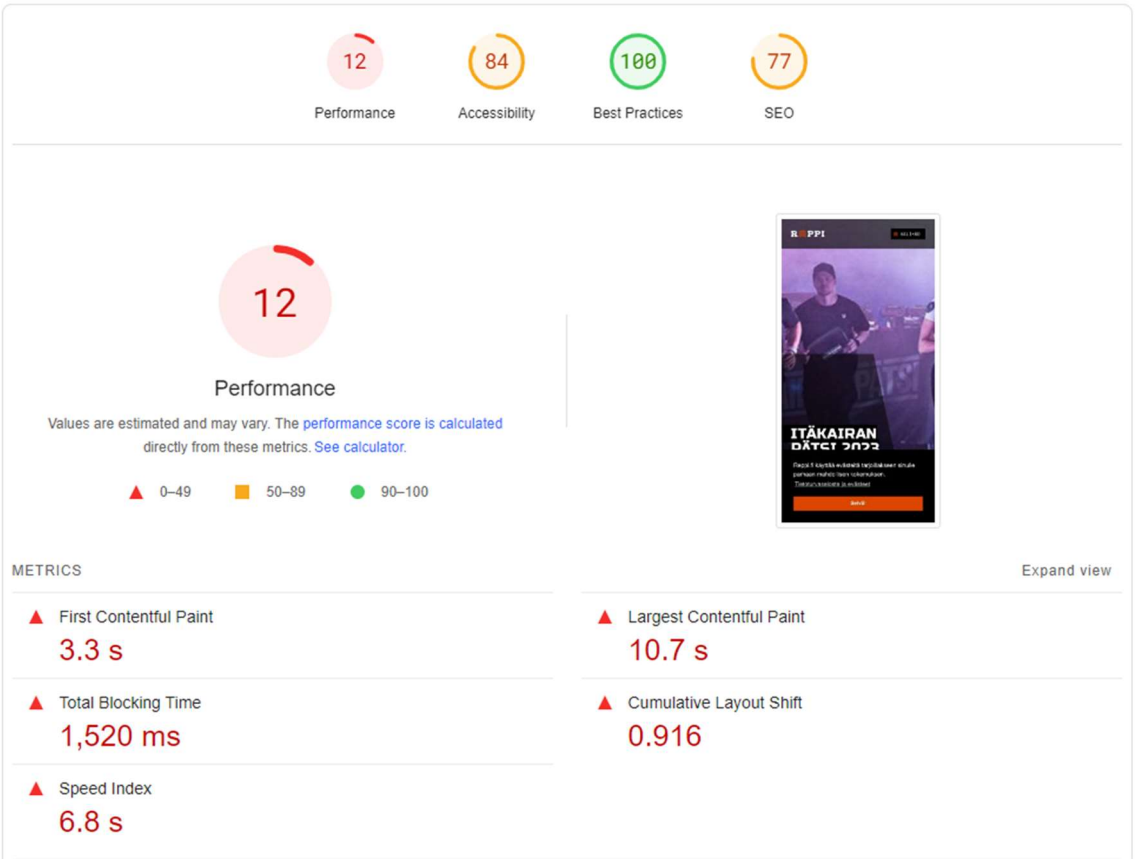
Kaikki edellä mainitut mittarit esittivät suhteellisen tärkeää roolia projektissa sillä ne ovat konkreettisia mittayksiköitä, joilla pystyy vertailemaan sivuja keskenään. Käyn seuraavaksi läpi mittaustulokset ensin vanhasta sivusta ja sen jälkeen uudesta.

Vanhasta sivusta(kuva 10) voimme huomata hälyttävän suuren CLS arvon, joka johtuu aikaisemmin selitetystä tilanteessa, jossa staattinen sisältö latautuu välittömästi ja vasta usean sekunnin jälkeen loppusisältö ilmestyy sivulle puskien koko sivua alaspäin. SEO pisteissä olisi myös parantamisen varaa, vaikka noin 70 pisteen arvo ei ole huono.

FCP arvo on taas hyvä, sillä suuri osa staattisesta sisällöstä latautuu välittömästi. Varsinkin mobiililla (kuva 11) voimme huomata CLS:n sekä LCP:n suuret lukemat. Suuretkaan luvut eivät silti tarkoita, että sivusto olisi huono vaan enemmän sitä, että parantamisen varaa löytyy.



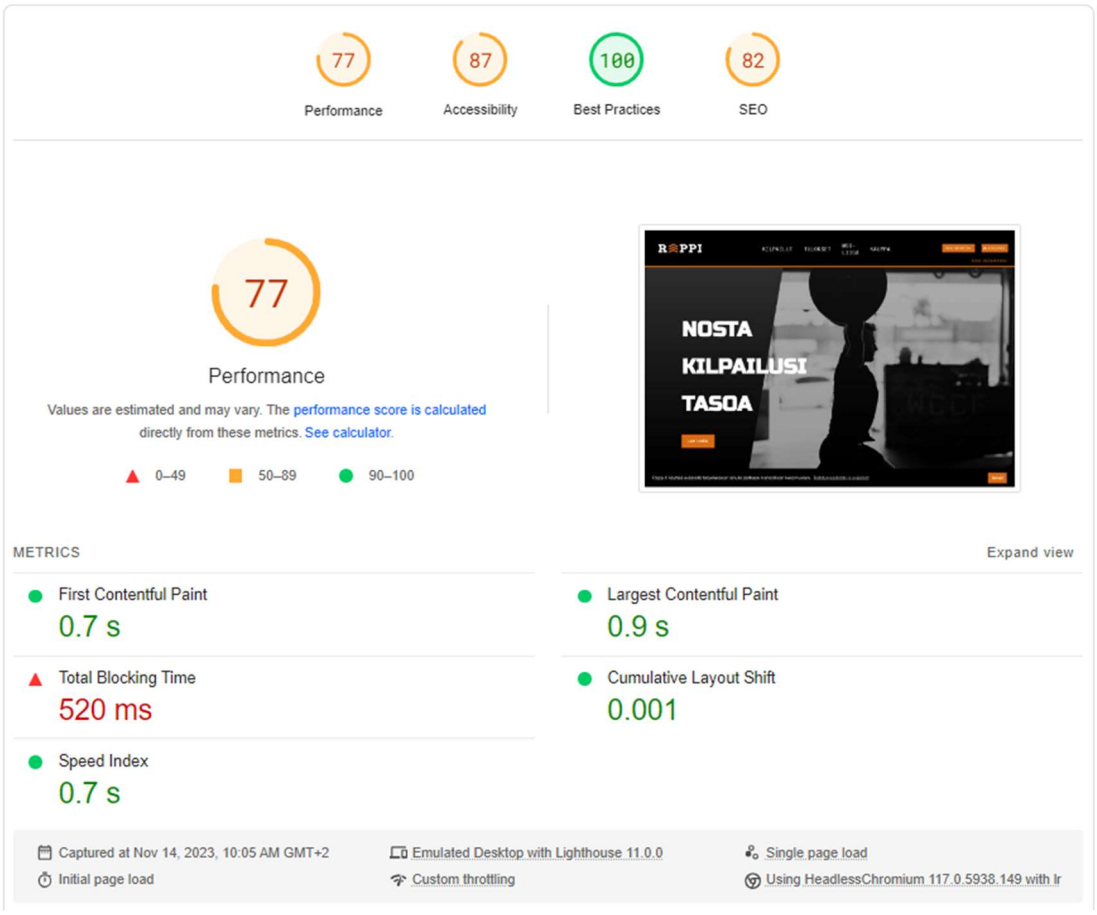
Kuva 9 Reppi vanhan etusivun tulokset



Kuva 10 Reppi vanhan etusivun tulokset mobiilissa

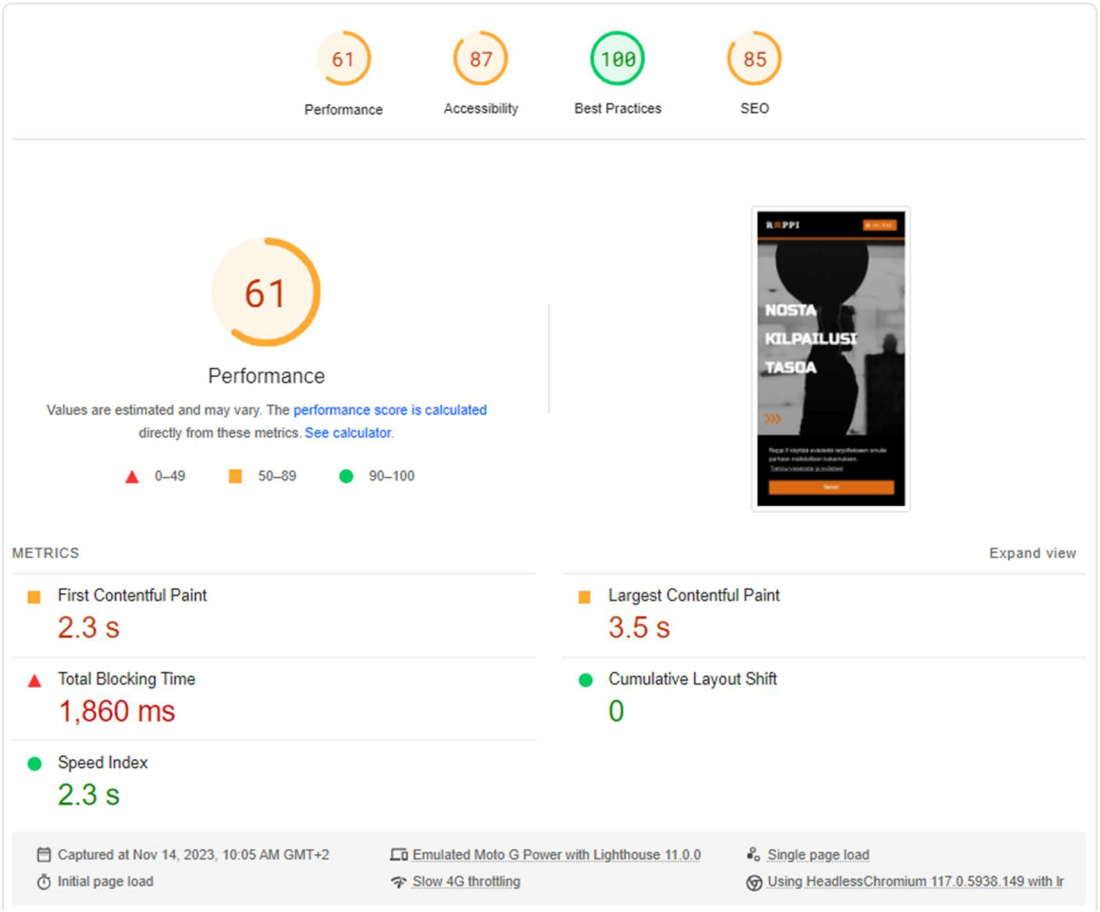
Uudessa sivussa (kuva 12) taas voimme huomata, että SEO arvo on kohonnut huomattavasti, sekä CLS arvo on käytännössä 0, eli ideaali. Parannuksia on tapahtunut lähtökohtaisesti kaikissa pisteissä.

TBT arvon kasvu johtuu osin siitä, että uudella etusivulla on huomattavasti suurempi määrä dataa ja laskuteknisiä asioita kuin vanhalla etusivulla. Esimerkiksi kilpailukalenteri on jokseenkin laskentaintensiivinen, sillä jokaisen kisakalenteriobjektin kohdalla pitää tehdä ainakin seuraavat asiat: Tarkasta onko kisa käynnissä, onko ilmoittautuminen auki ja onko kisalla linkattu kisaobjekti. Jos kisan ilmoittautuminen on avoinna, piirrä nappula josta kisaan pääsee ilmoittautumaan. Jos kisa on online-kisa, piirretään nappula tulosten syöttämiselle. Jos kisa on ohi, se suodatetaan omaan listaansa, eli menneisiin kisoihin. Edellä mainittuja asioita ei tarkasteta, jos kisa on jo ohi. Nämä kaikki laskentatekniset asiat vaikuttavat omalta osaltaan TBT:hen.



Kuva 11 Reppi uuden etusivun tulokset

On tärkeää huomata, että mittaustuloksissa (varsinkin mobiilissa (kuva 13)) verkon nopeutta rajoitetaan keinotekoisesti hitaammaksi, joka saattaa huonontaa pisteytystä SSR renderöinnin kannalta. Tämän takia FCP, TBT sekä LCP tulokset voivat näyttää huonolta mobiilissa, vaikka reaalimaailman tilanteessa sivu latautuisikin paljon nopeammin.



Kuva 12 uuden etusivun tulokset mobiilissa

Mutta jopa näissä tuloksissa huomataan, että olemme käytännössä poistaneet sivuston CLS:n ja parantaneet jo näin käyttäjäkokemusta. SEO pisteytys on myös noussut SSR:n myötä sillä koko sivu palautetaan indeksointia tekevälle hakukoneelle heti. Tällä tapaa sen ei tarvitse odottaa/suorittaa JavaScriptiä, olettaen että se osaa edes avata sivun tai suorittaa JavaScriptiä. Tarvittavia tietoja sivun indeksointiin hakukoneita varten on jo HTML tiedostossa valmiina, eikä kaikkea tarvitse rakentaa siihen tyhjästä, niin kuin tavallisessa SPA-sovelluksessa.

6 YHTEENVETO

Tässä opinnäytetyössä tutustuttiin SSR- ja SPA-verkkosovellusten toimintaan ja SSR-tekniikan implementointiin olemassa olevaan SPA-sovellukseen. Myös verkkosivujen toiminnan pisteyttämiseen ja arviointiin on paneuduttu huomattavasti.

Opinnäytetyön tarkoitus ei ole olla oppaana SSR-sovelluksen kehittämiseen vaan demonstroida yhtä tapaa monesta markkinoilla olevasta vaihtoehdosta ja esittää konkreettisia mittaustuloksia ja hyötyjä.

Opinnäytetyön toiminnallinen toteutus tuli kokonaisuudessaan toimeksiantajalle tuotantokäyttöön. Projektin päätyttyä toimeksiantajalla on käytössään uusi nopea etusivu Reppi.fi palvelulle, jota on helppo ylläpitää ja päivittää myös ilman sovelluskehittäjää.

Projektin aikana opin paljon Serverless-toteutuksista ja erityisesti Lambdan toiminnasta ja sen kriittisyydestä tämänkaltaisissa toteutuksissa. Aihe oli erittäin mielenkiintoinen ja uskon vahvasti, että siitä on minulle hyötyä tulevaisuudessa. Tietysti pitää ottaa huomioon, että Serverless-maailmassa evoluutio on nopeaa ja tapa miten asiat tehdään tänään voi olla erilainen jo huomenna.

Projektin päättyessä olimme kokonaisvaltaisesti tyytyväisiä mittaustuloksiin sekä sivuston toimintaan, jota on miellyttävä katsella, käyttää sekä ylläpitää.

LÄHTEET

Adobe Experience Cloud Team. (18.08.2023a) [A guide to traditional vs. single-page apps (SPAs)] https://business.adobe.com/blog/basics/differences-between-traditional-web-apps-and-single-page-apps#what-is-a-traditional-web-application

Adobe Experience Cloud Team. (19.07.2023b) [Single-page applications (SPAs) - what they are and how they work] https://business.adobe.com/blog/basics/learn-the-benefits-of-single-page-apps-spa

Algolia. (n.d.) Haettu 30.11.2023 osoitteesta https://www.algolia.com/

Amazon. (n.d.-a) Haettu 19.1.2024 osoitteesta https://aws.amazon.com/cloudfront/

Amazon. (n.d.-b) Haettu 12.10.2023 osoitteesta https://aws.amazon.com/cloudfront/features/

Amazon. (n.d.-c) Haettu 12.10.2023 osoitteesta https://aws.amazon.com/s3/

Amazon. (n.d.-d) Haettu 12.10.2023 osoitteesta https://docs.aws.amazon.com/lambda/latest/dg/welcome.html

Amazon. (n.d.-e) Haettu 12.10.2023 osoitteesta https://aws.amazon.com/api-gateway/

Angular. (n.d.-a) Haettu 24.11.2023 osoitteesta https://angular.io/errors/NG0506

Angular. (n.d.-b) [Why use SSR?] Haettu 11.11.2023 osoitteesta https://angular.io/guide/ssr

Directus. (n.d.) Haettu 10.10.2023 osoitteesta https://directus.io

Freeman, A. (2022). Pro Angular: Build powerful and dynamic web apps (Fifth edition.). Apress L. P. https://doi.org/10.1007/978-1-4842-8176-5

Google Developers. (11.05.2022a) [Total Blocking Time] https://web.dev/tbt/

Google Developers. (19.10.2022b). [First Contentful Paint] https://web.dev/fcp/

Google Developers. (19.10.2022c). [Largest Contentful Paint] https://web.dev/lcp/

Google Developers. (12.01.2023a). [Core Web Vitals] https://developers.google.com/search/docs/appearance/page-experience

Google Developers. (23.05.2023b). [Google Search Essentials] https://developers.google.com/search/docs/essentials?visit_id=638319149060925928-1085454061&rd=1

Google Developers. (14.09.2023c) [An image format for the Web] https://developers.google.com/speed/webp

Google Developers. (27.09.2023d). [Get started with Search: a developer's guide] https://developers.google.com/search/docs/fundamentals/get-started-developers

Google Developers. (10.11.2023e). osoitteesta https://developers.google.com/search/docs/crawling-indexing/javascript/dynamic-rendering

Google Developers. (05.02.2024a). [Cumulative Layout Shift] https://web.dev/cls/

Google Developers. (05.02.2024b). [Cumulative Layout Shift] https://web.dev/cls/

Jeff Posnick. (18.07.2019) https://web.dev/articles/stale-while-revalidate

Jithin Jude Paul. (2023). Distributed serverless architecture on aws. https://doi.org/10.1007/978-1-4842-9159-7

Kristi Hines, Searchenginejournal.com (10.18.2021). [Core Web Vitals as a Google Ranking Factor: What You Need to Know] https://www.searchenginejournal.com/ranking-factors/core-web-vitals/

Liron Tagger, Medium. (08.05.2023). [Using Angular with WebWorkers] https://medium.com/palo-alto-networks-cortex-dev/using-angular-with-web-workers-33ea60bb3efc

Oka Yutaka. (04.03.2021). https://aws.amazon.com/blogs/networking-and-content-delivery/improve-your-website-availability-with-amazon-cloudfront/

Red Hat. 05.11.2022 [What is serverless?] https://www.redhat.com/en/topics/cloud-native-apps/what-is-serverless

Reppi. (06.05.2021). Haettu 10.10.2023 osoitteesta https://reppi.fi/about/reppi

Stack Overflow. (2023). Haettu 12.12.2023 osoitteesta: https://survey.stackoverflow.co/2023/#most-popular-technologies-webframe

Typescript. (28.11.2023) https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html

Tim Davidson. 28.02.2023 https://cleancommit.io/blog/spa-vs-mpa-which-is-the-king/