Ville Hytönen

Moderni verkkokauppa

Metropolia Ammattikorkeakoulu
Insinööri (AMK)
Tieto- ja viestintätekniikka
Insinöörityö
7.11.2022

Tiivistelmä

Tekijä:	Ville Hytönen
Otsikko:	Moderni verkkokauppa
Sivumäärä:	24 sivua
Aika:	7.11.2022
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Ohjelmistotuotanto
Ohjaajat:	Lehtori Simo Silander

Insinöörityössä perehdyttiin verkkokaupan tekemiseen Gatsby Stripen ja Tailwind CSS:n kanssa. Verkkokaupan hostaus tapahtuu Netlifyn CDN-palvelussa ja siinä tulee olemaan automaattinen, kun sitä päivitetään Git repositoryn tai Stripen tuotteiden päivittäessä. Verkkokaupan valikoimaan voidaan vaikuttaa ilman, että tarvitaan kehittäjän apua.

Verkkokauppa käyttää Jamstack-arkkitehtuuria, jossa luodaan staattisia verkkosivuja, joiden hostaus tapahtuu CDN-palvelussa ja ne yhdistyvät sieltä Stripe-palveluihin, jotka ottavat maksut vastaan ja tämän jälkeen ohjaavat takaisin verkkokauppaan. Verkkokauppaan luodaan tyylit Tailwind-kirjastoa käyttäen.

Työn tuloksena saatiin toimiva verkkokauppa, jonka jatkuvat kustannukset ovat erittäin alhaiset. Tämän takia verkkokauppa soveltuu juuri sopivasti sen tarkoitukseen.

Stripen palveluiden opiskelemisen jälkeen tässä ei ilmennyt mitään pahempia ongelmia. Loppujen lopuksi verkkokauppa ajaa tarvittavat hommansa tarkoitusta varten, mutta parannettavaa löytyy myös.

Avainsanat: Gatsby, Stripe,Tailwind CSS, Jamstack

Abstract

Author:	Ville Hytönen
Title:	Modern E-Commerce Site.
Number of Pages:	24 pages
Date:	7 November 2022
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Professional Major:	Software Engineering
Supervisors:	Simo Silander, Senior Lecturer

The thesis focuses on creating an e-commerce site with Gatsby, Stripe and Tailwind CSS. THE e-commerce site is hosted at Netflify's Cloud distribution network, and it has continuous integration and continuous delivery from the git repository and Stripe hook. That way the manager of the e-commerce site can update the stock without a developer.

The site uses Jamstack architecture, where one creates a static website with Gatsby, and they are hosted at CDN. The Static website is able connect with Stripe Service enabling making payments. Styling of the e-commerce site is made with Tailwind.

The result of the study, is a working e-commerce site that has low maintenance costs, thereby meeting initial requirements well.

Keywords: Gatsby, Stripe, Tailwind CSS, Jamstack

Sisällys

Lyhenteet

API: *Application programming interface*, ohjelmointirajapinta on määritelmä, jonka mukaan eri ohjelmat voivat tehdä pyyntöjä ja vaihtaa tietoja eli keskustella keskenään.

CI/CD: Tulee englannin kielen sanoista *continuous integration* ja *continuous delivery*. Tarkoittaa jatkuvaa integrointia ja julkaisua.

CDN: Sisällönjakeluverkko tai sisällönjakoverkko on maantieteellisesti hajautettu verkko, jonka tarkoitus on palvelun tarjoaminen loppukäyttäjien lähellä paremmalla toimitusvarmuudella ja pienellä viiveellä.

HTML: *Hypertext Markup Language*. Se on kuvauskieli, jonka avulla kuvataan nettisivun rakenne sekä sivun sisältö.

Jamstack: Tulee sanoista Javascript, API ja Markup. On arkkitehtuuri suunniteltu tekemään internetistä nopeampi, turvallisempi ja skaalattavampi.

Loader: Webpacking käyttämä moduulin latausohjelma.

MERN: MERN-stack tarkoittaa ohjelmistokokonaisuutta, joka rakentuu teknologioista MongoDB, Express.js, React.js ja Node.js.

Plugin: Webpackin ja Gatsbyn käyttämä koodinpätkä, joka parantaa sen toiminnallisuutta.

Reducer: Redux-arkkitehtuurin käyttämä rajapinta.

SEO: *Search engine optimization* eli hakukoneoptimointi.

Serverless: On pilvelle natiivi kehitysmalli, joka sallii kehittää ja ylläpitää sovelluksia ilman, että täytyy hallita palvelimia.

URL: *Uniform Resource Identifier* on merkkijono, jolla kerrotaan tietyn tiedon paikka.

1 Johdanto

Insinöörityön tarkoituksena on luoda verkkokauppa, jossa olisi mahdollisimman vähän jatkuvia kustannuksia sekä katsoa, miten staattisen verkkosivun pystyy valjastamaan verkkokauppakäyttöön, missä käyttäjän pitää voida lisätä ja poistaa tuotteita verkkokaupasta. Verkkokaupan teen tutuilleni, joilla on omaehtoinen levy-yhtiö ja pari bändiä. Tavoitteena on luoda mahdollisimman vähän jatkuvia kuluja ja vain katsoa, tulisiko tilauksia, joita ei muuten tulisi. Verkkokaupan ei tarvitse olla aivan viimeiseen asti hiottu, mutta jos se lisää bändin oheistuotteiden myyntiä, niin tätä voidaan päivittää paremmaksi.

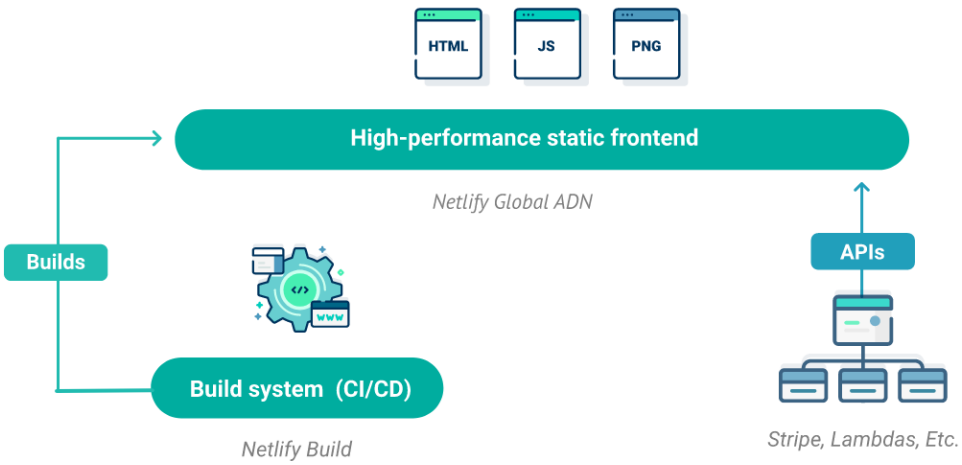
Insinöörityössä ensiksi perehdytään oleellisiin teknologioihin ja palveluihin, joita työssä käytetään. Tämä sisältää isompia aiheita kuten Jamstack, CI/CD ja hostaus. Tämän jälkeen käydään tarkemmin läpi teknologioita Gatsby ja Tailwind. Lopuksi käymme läpi tätä projektia varten tarvittavat Stripen palvelut.

Seuraavaksi insinöörityössä käydään läpi projektin aloitus ja rakenne. Tämän jälkeen syvennytään vähän tarkemmin, mitä komponentteja tarvitaan, jonka jälkeen perehdytään isompiin kokonaisuuksiin.

2 Teknologiat

2.1 Jamstack

Jamstack tulee termeistä JavaScript, API ja Markup [1]. Jamstack-teknologialla sivujen rakennusvaiheessa haetaan kaikki mahdollinen data, jonka jälkeen rakennusvaihe luo staattiset verkkosivut. Staattiset verkkosivut ovat yleensä turvallisempia, halvempia ja suorituskykyisempiä dynaamiseen verkkosivuun verrattuna. Jamstack erottuu muista stackeista, kuten MERN-stackistä erottamalla front endin backend-palveluista. [2.] Frontendin tämän jälkeen palvellaan käyttäjille CDN-palveluista, kuten Netlify tai AWS lambda. CDN-palveluntarjoajat tarjoavat yleensä erittäin skaalautuvaa, edullista ja turvallista palvelua.



Kuva 1. Jamstack-malli.

Kuvassa 1 on kaavio, miten Jamstack-sovellukset toimivat. Kuvassa CI/CD:n rakennusjärjestelmä saa tiedon päivityksestä ja käynnistää staattisen frontendin rakennuksen. Tässä tapauksessa se on Gatsbyn rakennusvaihe.

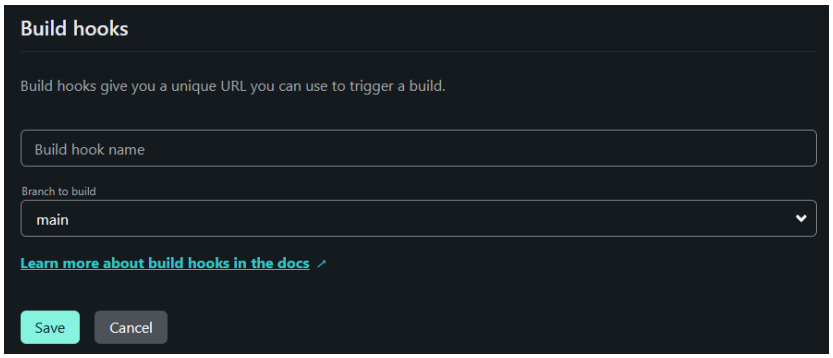
2.2 CI/CD

CI/CD kirjainyhdistelmällä tarkoitetaan jatkuvaa integraatiota ja jatkuvaa toimitusta. Sillä viitataan automatisoituun testien juoksuun, integraatioon sekä koodin

tuotantoonvientiin [3]. Tässä tapauksessa, kun kaupan tuotteiden tiedot haetaan rakennusvaiheessa, halutaan niiden päivittymisestä CI/CD-putkeen erikseen tehdä käynnistysmekanismi. Tämä käynnistää rakennusvaiheen ja tekee uuden staattisen verkkosivun kaikilla kaupan tuotteilla. Tämä luodaan tässä tapauksessa tekemällä URL-päätepiste Netlifyihin, joka käynnistää rakennusvaiheen.

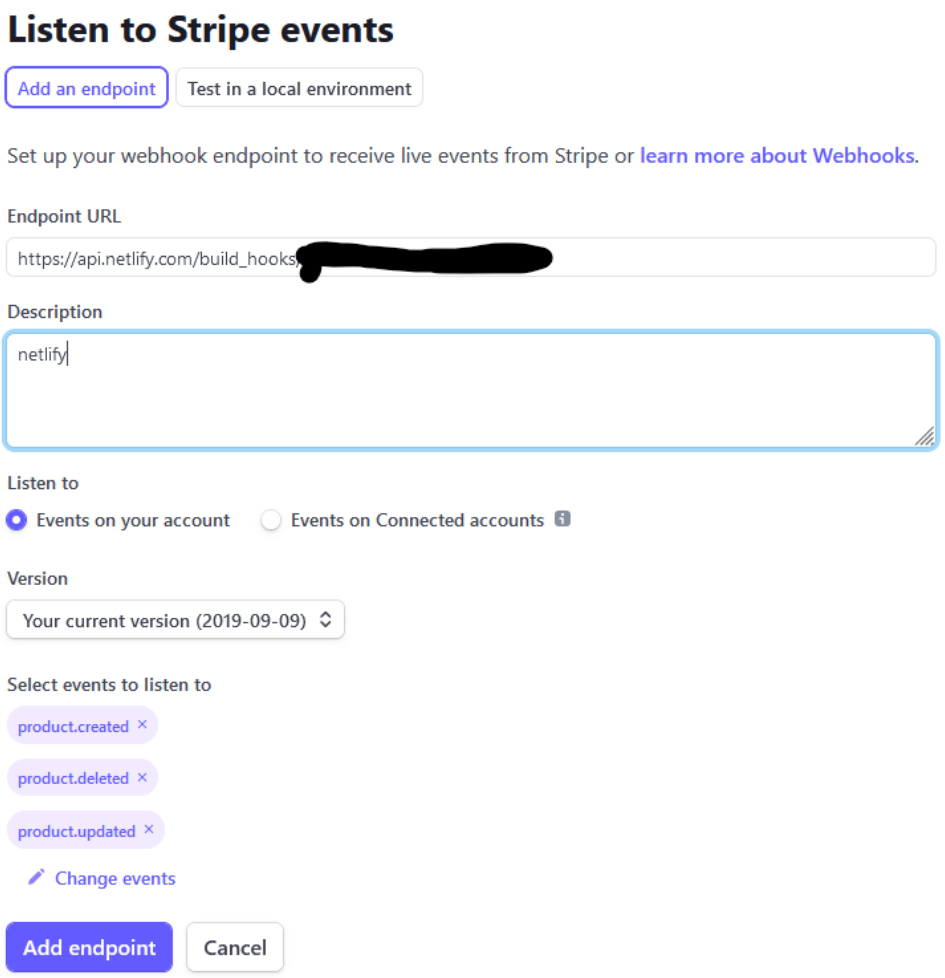
2.3 Hostaus

Jamstack suosii CDN-hostausta, ja valitsin sitä varten Netlifyn, koska siinä on riittävät palvelut sekä tarpeeksi kattava ilmainen aloittelijapaketti. Netlify on alusta, joka tarjoaa pilvihostausta ja serverless-arkkitehtuurilla olevia backend-toimintoja. CI/CD eli jatkuva integraatio ja jatkuva käyttöönotto on tehty erittäin helpoksi Netlifyllä. Kun koodi valitaan githubista niin se jää automaattisesti katsomaan main branchin päivityksiä. Haluttaessa tehdä lisää rakennuksen aloittavia toimia, kuten tässä tapauksessa Stripen päivitys, pitää vain antaa sille nimi, ja Netlify antaa ulos URL:in. Sen luominen onnistuu kuvassa 2 näkyvillä tiedoilla: sille täytyy vain antaa nimi ja valita git-oksa, joka on tässä tapauksessa main.



Kuva 2. Netlifyhin URL-luonti, jotta Stripe voi yhdistää siihen.

Seuraavaksi tulee Stripen puolelta vastaava (kuva 3), jossa on valittu tuotteiden päivitystapahtumat.



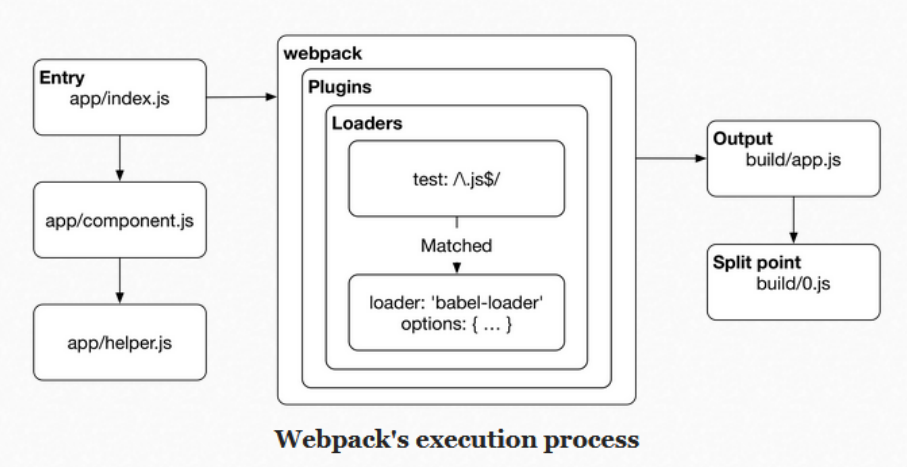
Kuva 3. Ylhäällä näkyy Netlifyn antama päivitys-URL ja alempana, mitkä tapahtumat lähettävät päivitystapahtuman.

2.4 Gatsby

Gatsby on avoimen lähdekoodin kehitysympäristö, joka yhdistää Reactin, GraphQL:n sekä Webpackin luodakseen staattisia nettisivuja. Sen tarkoituksena on ollut luoda kehittäjäystävällinen tapa luoda nopeita ja hyvin optimoituja sovelluksia ja verkkosivuja [4].

2.4.1 Webpack

Webpack on moduulirakentaja. Webpackissä pystyy optimoimaan JavaScriptiä, Typescriptiä, CSS:ää, kuvia sekä kaikkea tarvittavaa. Webpackiin kirjoitetaan webpack.config.js-tiedosto, johon laitetaan aloitustiedoston tai -tiedostojen sijainti, minne halutaan, että valmis tiedosto sijoitetaan. Tämän lisäksi, jos halutaessa Webpackin ymmärtävän esimerkiksi SASS- tai CSS-syntaksia joudutaan lisäämään loader:in kyseiselle kielelle ja kertomaan sen Webpackin asetuksille. Tämän lisäksi Webpack käyttää plugineja, joilla pystyy esimerkiksi lisäämään loki-ilmoituksia tai lisäämään HTML-mallipohjan projekteille [5]. Webpackistä tuli erityisen käytetty avoimen lähdekoodin kirjasto sen jälkeen, kun julkaistiin HMR (Hot Module Replacement) ja React [6]. Tällöin Reactia pystyi kehittämään ilman kehitysympäristön uudelleen käynnistämistä aina muutoksen tapahduttua JavaScript-tiedostossa. Webpackin toiminta siis perustuu aloitus-tiedoston, joka on index.js kuvan tapauksessa, lukemiseen, sen käsittelemiseen halutulla määritellyllä loader:eilla ja tämän lisäksi toiminnallisuuksia voidaan lisätä plugineilla. Ne voivat toimia ennen tai jälkeen loadereita (kuva 4).



Kuva 4. Webpackin toimintakaavio.

2.4.2 GraphQL

GraphQL on kyselykieli kuten SQL, mutta siinä keskitytään antamaan asiakasohjelmaan juuri sen, mitä tarvitsee. GraphQL kehitettiin olemaan nopea, joustava ja kehittäjäystävällinen. GraphQL käyttää schemaa kuvaamaan kaikkea GraphQL:stä haettavaan tietoa. Gatsbyn rakennusvaiheessa se juoksee kaikki projektiin kytketyt pluginit ja luo sen pohjalta GraphQL-scheman. Se näyttää seuraavalta (kuva 5). Stripen plugin tuo kuvaan StripePricen ja allStripePricen.

Kuva 5. GraphQL:stä löytyvät tiedot, josta kehittäjä voi päättää, mitkä tiedot pitää hakea mihinkin komponenttiin tai sivuun.

Kun kysely tulee sisään, GraphQL varmistaa sen schemaa vasten ja sitten suorittaa validoidut kyselyt. Esimerkkikoodi 1 on projektissa käytetty Stripe-tietojen hakukysely.

```
query ProductPrices {
        prices: allStripePrice(
          filter: { active: { eq: true }, currency: { eq: "eur" } }
          sort: { fields: [unit_amount] }
        ) {
          edges {
            node {
```

```
            id
            active
            currency
            unit_amount
            product {
              id
              name
              images
            }
          }
        }
      }
    }
`
```

Esimerkkikoodi 1. GraphQL-kysely, joka hakee kaikki tuotetiedot.

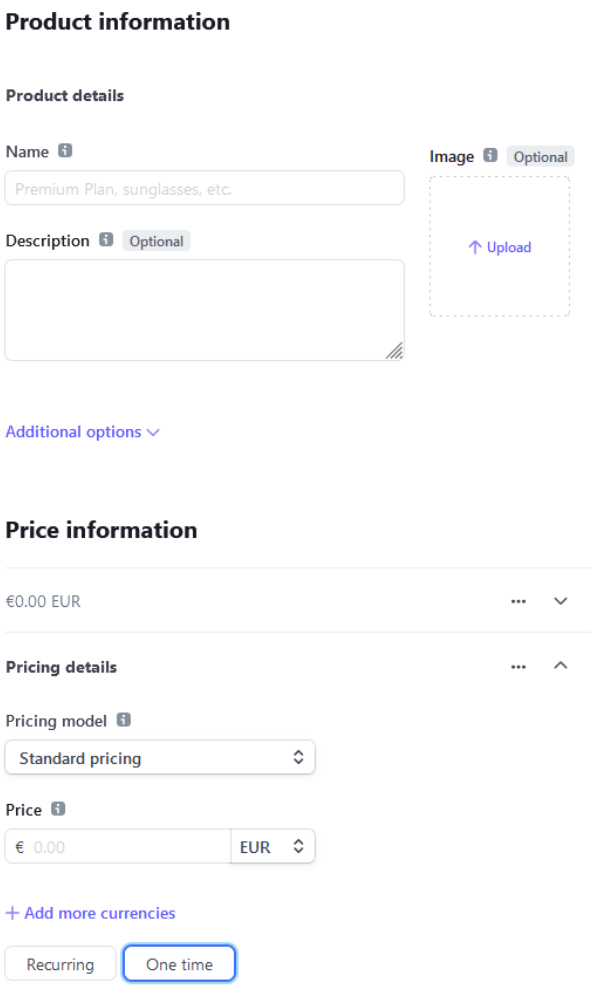
Resolveri on funktio, joka on vastuussa data tuonnista yhteen scheman kenttään [7]. GraphQL hakee tarvittavat tiedot resolverillä, jotka pystyvät hakemaan tarvittavat tiedot tietokannasta tai API:sta. GraphQL on pyritty kehittämään tyylillä, jolla olisi mahdollisimman helppo integroida monta tietokantaa tai API:a.

Gatsbyyn lisätään paljon toiminnallisuuksia pluginien kanssa, joista yleisimmät hakevat dataa GraphQL-kyselyitä varten. Gatsbyssä on myös sisäänrakennettu reach-router, joka tekee siitä single page applicationin ja se automaattisesti jakaa koodia paloihin Webpackin kanssa suorituskyvyn parantamiseksi. Tässä projektissa käytän esim. Gatsby-source-stripe pluginia, joka hakee Stripen API:sta rakennusvaiheessa tuotteiden tiedot, minkä jälkeen ne menevät automaattisesti GraphQL-schemaan kuvassa 5 näkyvän “StripePrice” paikalle. Tämän lisäksi tässä projektissa on pari kuvan optimointi-pluginia ja yksi SEO-plugin.

2.5 Stripe

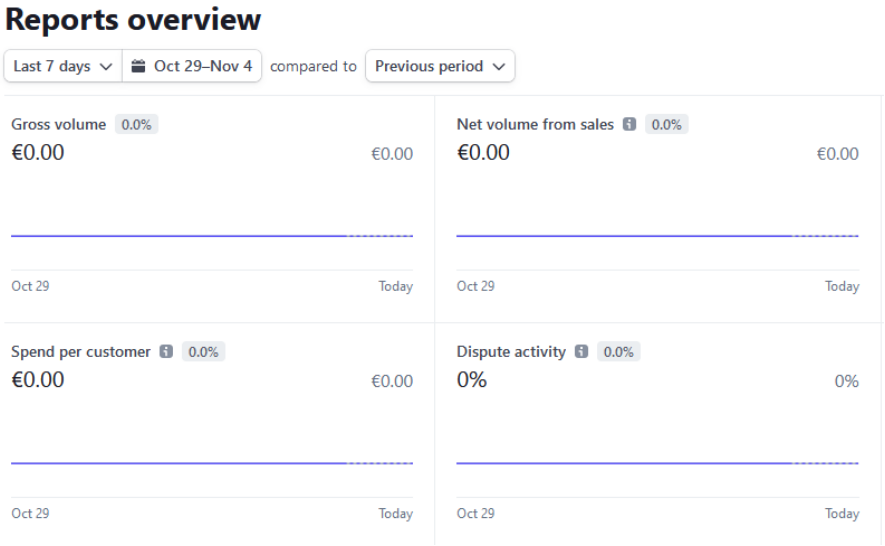
Stripe on palveluntarjoaja maksujen käsittelyyn sekä verkkokauppapuolelle. Stripen erottaa useasta muusta palveluntarjoajasta sen hinnoittelu, jossa maksetaan pieni osuus jokaisesta yksittäisestä maksusta kuukausimaksun sijaan.

Esimerkiksi Euroopan talousalueen maksukorteista tulee 1,4 % + 25 snt per käsitelty maksu. Jos tätä vertaa kilpailijoihin kuten Wixiin, Shopifyhin, BigCommerceen sekä SquareSpaceen, joiden kuukausi maksut ovat 25 – 30 € välillä, tarvitaan Stripessä yli tuhannen euron kuukausimyynti [8].



Kuva 6. Stripe-tuotteen lisäyslomake.

Stripen käyttöliittymästä voi käydä helposti lisätä uusia tuotteita. Verkkokaupaan pitää vain syöttää pakolliset tiedot nimi ja hinta (kuva 6) tai tehdä kampanjakoodeja tai säätää postituskuluja.



Kuva 7. Osa Stripen bisnestatistiikkasivusta.

Sieltä näkee selkeästi myös kaikki uudet asiakkaat, myynnin määrät, yksittäiset myynnit sekä verot. Osan näistä otin näytille Stripen tarjoamaan bisnestatistiikkaan (kuva 7). Käyttäjän ja businespuolen lisäksi löytyy developer-puolelta jotain ominaisuuksia, joista saa API-avaimet, lokitiedot ja tapahtumat.

2.6 Tailwind CSS

Tailwind on lähdekoodin CSS-työkalu. Siinä on paljon (kuva 8) pieniä utiliteettiluokkia, jotka pitää asettaa HTML-komponenttien luokkaan, jotta komponentit saavat halutun tyylin. [9.] Tarvittaessa omat kustomoidut väriteemat voidaan syöttää tailwind.config.js-tiedostoon. Kehitysvaiheessa Tailwind lisää ison CSS-tiedoston, jossa on kaikki utiliteetti-luokat, mutta rakennusvaiheessa kaikki tar-

vittavat CSS-luokat käydään läpi ja vain ne, jotka ovat tarvittavia, jäävät lopulliseen CSS-tiedostoon.

Kuva 8. Yläreunassa näkyy renderöity HTML-elementti ja sen alapuolella näkyy, miten se kirjoitetaan. Alimpana kuvassa näkyy, mitä Tailwind-luokat sisältävät.

2.7 Npm

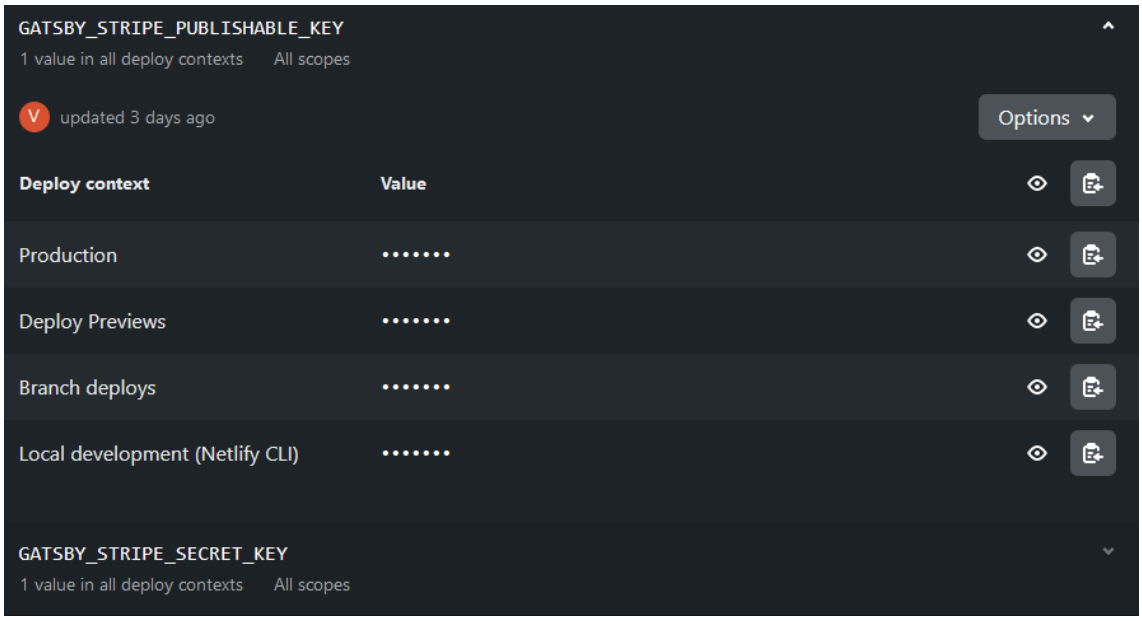
Npm on Node.js:n mukana tuleva paketinhallintajärjestelmä. Npm:ää käytetään pääasiallisesti avoimen lähdekoodin jakamiseen, lataamiseen ja päivittämiseen. Tässä projektissa käytetyt paketit liittyivät pääasiallisesti Gatsbyyn ja Reactiin. Lisätyt paketit voidaan tarkistaa package.json-tiedostosta ja asentaa helposti uuteen kehitysympäristöön.

3 Verkkosivu

3.1 Aloitus

Aloitin projektin luomalla tyhjän Gatsby-projektin käyttämällä komennon npm init gatsby [10]. Tämän jälkeen selvitin, mitkä paketit Stripe tarvitsee toimiakseen ja lisäsin niiden lisäksi myös postcss- ja tailwind-paketit.

Tämä jälkeen lisäsin ympäristömuuttujat kuten Stripen avaimet ja URL-arvon projektin ympäristömuuttuja-tiedostoon ja Netlifyn-palvelimelle. URL-arvoa ei tarvitse määritellä Netlifyyn, kun se tapahtuu itsestään. Kuvasta 9 näkee Stripeen käsin lisätyt arvot.



Kuva 9. Netlifyn sivulta löytyvät ympäristömuuttujat.

3.2 Projektin rakenne

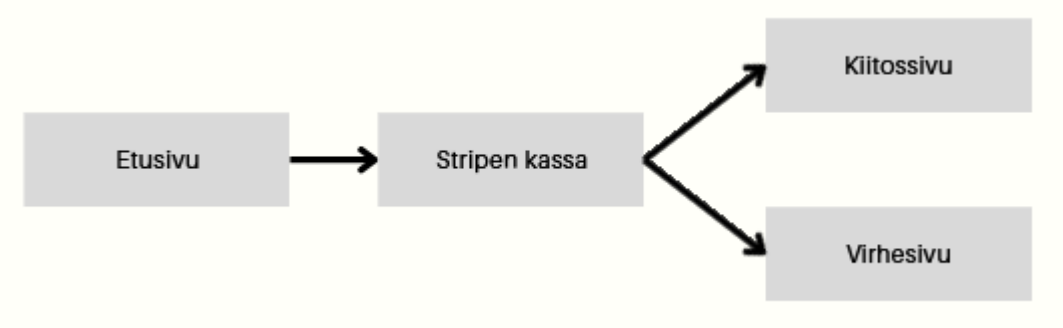
Projektissa on Gatsby-pohjan tiedostorakenne (kuva 10), johon on lisätty actions- ja reducers-kansiot ostoskoria varten. Jokaiselle eri pages-kansiossa olevalle sivulle tulee eri JavaScript- ja HTML-tiedosto, mikä nopeuttaa käyttäjien sivujen lataamista ja renderöintiä.

Kuva 10. Tiedosto- ja kansiorakenne.

Actions-kansio sisältää vain reducerin käyttämiä toimintojen nimiä. Kirjoittamalla niistä muuttujia ja käyttämällä niitä paikoissa, joissa tarvitsee, pyritään selkeyttämään käytettäviä toimintoja, jos niitä olisi enemmän tai käytössä olisi useampi reducer. Pages-kansio sisältää seuraavat sivut.

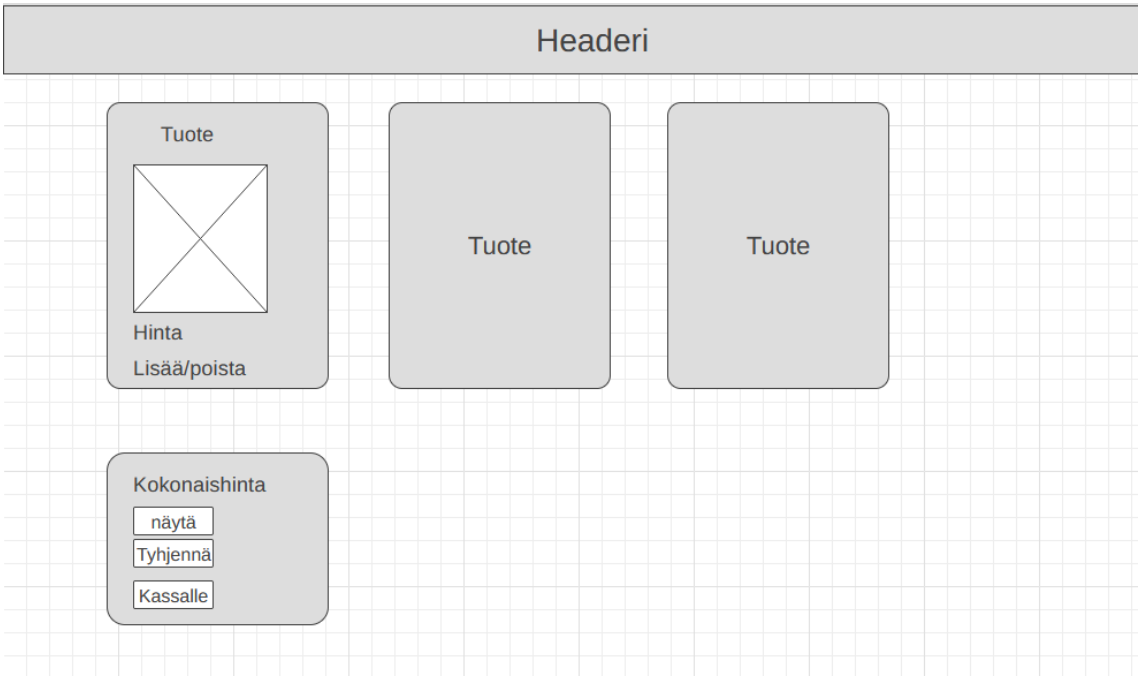
- 404
- index
- kiitos
- virhe.

404-virhesivu tulee esille silloin, kun käyttäjä laittaa URL-osoitteeseen jonkun jatkotunnisteen, joka ei ole esimerkiksi “verkkokauppa.com/pena”. Index-sivu tulee esiin, jos haettaessa verkkosivun vastaavalla URL-osoitteella “verkkokauppa.com”. Kiitos- ja virhesivut on lisätty Stripen kassaa varten. Sieltä ohjataan onnistuessa kiitossivulle tai sitten, jos jokin menee pieleen, virhesivulle. Navigointi toimii siis kokonaisuudessaan kuvan 11 osoittamalla tavalla.



Kuva 11. Mahdolliset verkkokaupan siirtymät.

Etusivun asettelu on aika yksinkertainen (kuva 12). Siinä on Headeri, myytävät tuotteet ja ostoskori.



Kuva 12. Verkkokaupan rautalankamalli.

3.3 Komponentit

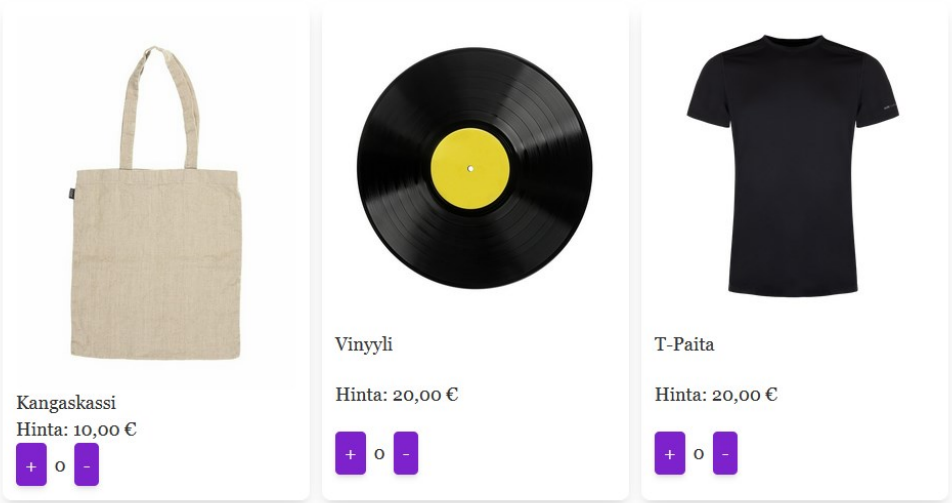
Projekti sisältää komponentit seuraaville asioille:

- myytävät tuotteet
- tuote
- ostoskori
- ostoskorintuote
- headeri
- sommittelu (layout)
- ohjaa kassalle
- SEO.

Myytävät tuotteet pitävät sisällään GraphQL-kyselyn, joka hakee tarvittavat tiedot kaikista tuotteista. Tämän lisäksi siinä on HTML-elementti, joka tekee tarvittavat tyylit tuotelistaa varten.

```
render={({ prices }) => (
      <div className="flex flex-row flex-wrap justify-between py-4">
        {prices.edges.map(({ node: price }) => {
          const newSku = {
            sku: price.id,
            name: price.product.name,
            price: price.unit_amount,
            currency: price.currency,
            image: price.product.images,
          }
          return <SkuCard key={price.id} sku={newSku} dispatch={dis-
patch} />
        })}
      </div>
```

Esimerkkikoodi 2. Listan tyylittely HTML-elementti ja itse tuotelista.



Kuva 13. Miltä tuotteet näyttävät kehittämistä varten tehdyillä esimerkkituotteilla.

Tuote-komponentti on yksittäistä tuotetta varten tehty komponentti, jossa määritellään tuotteen ulkoasu ja mahdollisuudesta vaikuttaa tuotteiden määrään ostoskorissa. Kuvassa 13 näkyvät myytävät tuotteet.

Ostoskori-komponentti kertoo korissa olevien tuotteiden määrän, hinnan, antaa mahdollisuuden tyhjentää ostoskorin ja lisäksi käyttäjän halutessa renderöi kaikki valitut tuotteet.

Ostoskorin tuotekomponentin sisällä määritellään sen ulkoasu, jonka lisäksi annetaan mahdollisuus määrän muuttamiselle tai poistamiselle.

Headeri-komponentti on sivun yläreunassa oleva osio, jossa on sivun otsikko ja mahdollisuus palata etusivulle painamalla. Jos painaa kuvan 14. tekstin kohtaa pääse etusivulle.

Kuva 14. Sivun yläreunassa oleva headerin ulkoasu.

Sommittelu-komponentti vastaa koko sivun yhteisestä teemasta. Kaikki ulkoasuun liittyvät asiat, jotka toistuvat jokaisella sivulla, laitetaan tänne.

```
const Layout = ({ children }) => {
  const data = useStaticQuery(graphql`
    query SiteTitleQuery {
      site {
        siteMetadata {
          title
        }
      }
    }
  `)
  return (
    <>
      <Header siteTitle={data.site.siteMetadata?.title} />
      <div className="mx-auto max-w-[960px] pt-0 px-4 pb-6">
        <main>{children}</main>
        <footer className="mt-8">
          © {new Date().getFullYear()}, Built with
          {` `}
          <a className="text-purple-800" href="https://www.gats-
byjs.com">
            Gatsby
```

```
          </a>
        </footer>
      </div>
    </>
  )
}
```

Esimerkkikoodi 3. Sommittelu-komponentissa haetaan GraphQL-kyselyn kanssa sivun otsikko ja se annetaan Headeri-komponentille.

Kassalle ohjaus-komponentissa (esimerkkikoodi 4 RedirectToCheckout) tarvitsee alustaa Stripe-yhteys ja tuoda ostoskorin sisältö, mutta muuten se on vain nappi, johon laitetaan haluttu ulkoasu.

```
import React from 'react'
import { loadStripe } from '@stripe/stripe-js'

function RedirectToCheckout({ cart = [] }) {
  const stripePromise = loadStripe(process.env.GATSBY_STRIPE_PUBLISHA-
BLE_KEY)
  const checkoutOptions = {
    mode: 'payment',
    lineItems: cart.map(item => ({ price: item.sku, quantity:
item.quantity })),
    successUrl: `${process.env.URL}/kiitos`,
    cancelUrl: `${process.env.URL}/virhe`,
  }
  const RedirectToCheckout = async () => {
    const stripe = await stripePromise
    const result = await stripe.redirectToCheckout(checkoutOptions)
  }

  return (
    <div>
      <button
        className="bg-purple-700 m-1 transition duration-200 text-
white px-4 py-2 rounded shadow-md hover:shadow-xl hover:-translate-y-1
active:-translate-y-0.5 active:shadow-lg"
        onClick={RedirectToCheckout}
      >
        Siirry kassalle
      </button>
    </div>
  )
}
```

```
export default RedirectToCheckout
```

Esimerkkikoodi 4. Koodissa tuodaan ensin Reactin ja Stripen oma JavaScript-moduuli, jonka jälkeen Stripe alustetaan ja sitten laitetaan se napin taakse.

Luetelmassa SEO-komponentissa luodaan komponentti, joka luo vain HTML-elementtejä, jotka helpottavat hakukoneiden optimointia esimerkiksi kielen kertominen.

3.4 Ostoskori

Ostoskorin tähän projektiin loin Reactin omalla useReducerilla, jonne minun piti kehittää 3 toimintaa. Tämän tyylinen tilanhallinta React:issa on lähtenyt Redux-kirjastosta, mutta muutama vuosi takaperin React mahdollisti sen käytön myös ilman Reduxia. Kehitetyt toiminnot olivat seuraavat:

- Lisää ostoskoriin.
- Poista ostoskorista.
- Päivitä ostoskori.

Tälle tilanhallinta-arkkitehtuurille on erittäin yleistä, että toiminnat luodaan switch-kytkimen sisään. Jokainen toiminto on eri kytkimen vaiheen takana, esimerkki koodi yhdestä.

```
case types.ADD_TO_CART:
      doesItemExist = false
      const newState = state.map(item => {
        if (item.sku === action.payload.sku) {
          item.quantity += 1
          doesItemExist = true
        }
        return item
      })
      if (doesItemExist) {
        return newState
      }
      return [...state, { ...action.payload, quantity: 1 }]
```

Esimerkkikoodi 5. Tuotteen lisäys tilanhallintaan.

Tämän jälkeen React-komponentille, jonka tarvitsee käyttää jotain näistä kolmesta toiminnasta, laitetaan tapahtumankuuntelija seuraavan koodiesimerkin näyttämällä tavalla.

```
const SkuCard = ({ sku, dispatch }) => {
  const [count, setCount] = useState(0)
  const increment = () => {
    setCount(count + 1)
    dispatch({ type: ADD_TO_CART, payload: sku })
  }
  const decrament = () => {
    if (count > 0) {
      setCount(count - 1)
      dispatch({ type: REMOVE_FROM_CART, payload: sku })
    }
  }
  return (
    <div className="flex flex-col justify-around items-start p-3 mb-4
shadow-lg bg-white rounded-lg max-w-[300px]">
      <img src={sku.image} alt={sku.name} />
      <h4>{sku.name}</h4>
      <p>
        Hinta:{' '}
        {new Intl.NumberFormat('fi-FI', {
          style: 'currency',
          currency: 'EUR',
        }).format(sku.price / 100)}
      </p>
      <button
        className="bg-purple-700 transition duration-200 text-white px-2
py-1 rounded shadow-md hover:shadow-xl hover:-translate-y-1 active:-
translate-y-0.5 active:shadow-lg"
        onClick={increment}
      >
        +
      </button>

      <p>{count}</p>
      <button
        className="bg-purple-700 transition duration-200 text-white px-2
py-1 rounded shadow-md hover:shadow-xl hover:-translate-y-1 active:-
translate-y-0.5 active:shadow-lg"
        onClick={decrament}
      >
        -
      </button>
```

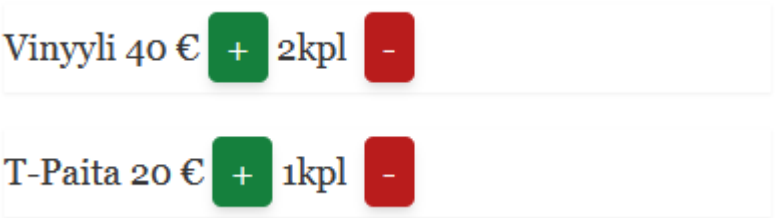
```
    </div>
  )
}
```

Esimerkkikoodi 6. Koodissa tehdään React-luokka, jonka jälkeen tehdään kappaleen lisäykselle ja poistamiselle valmiiksi kuuntelija. Sitten luodaan React-tyylillä HTML-elementit, joihin nämä laitetaan.

Ostoskorin ulkoasu on erittäin yksinkertainen (kuva 15) ja siinä on panostettu enemmän toiminallisuuteen.

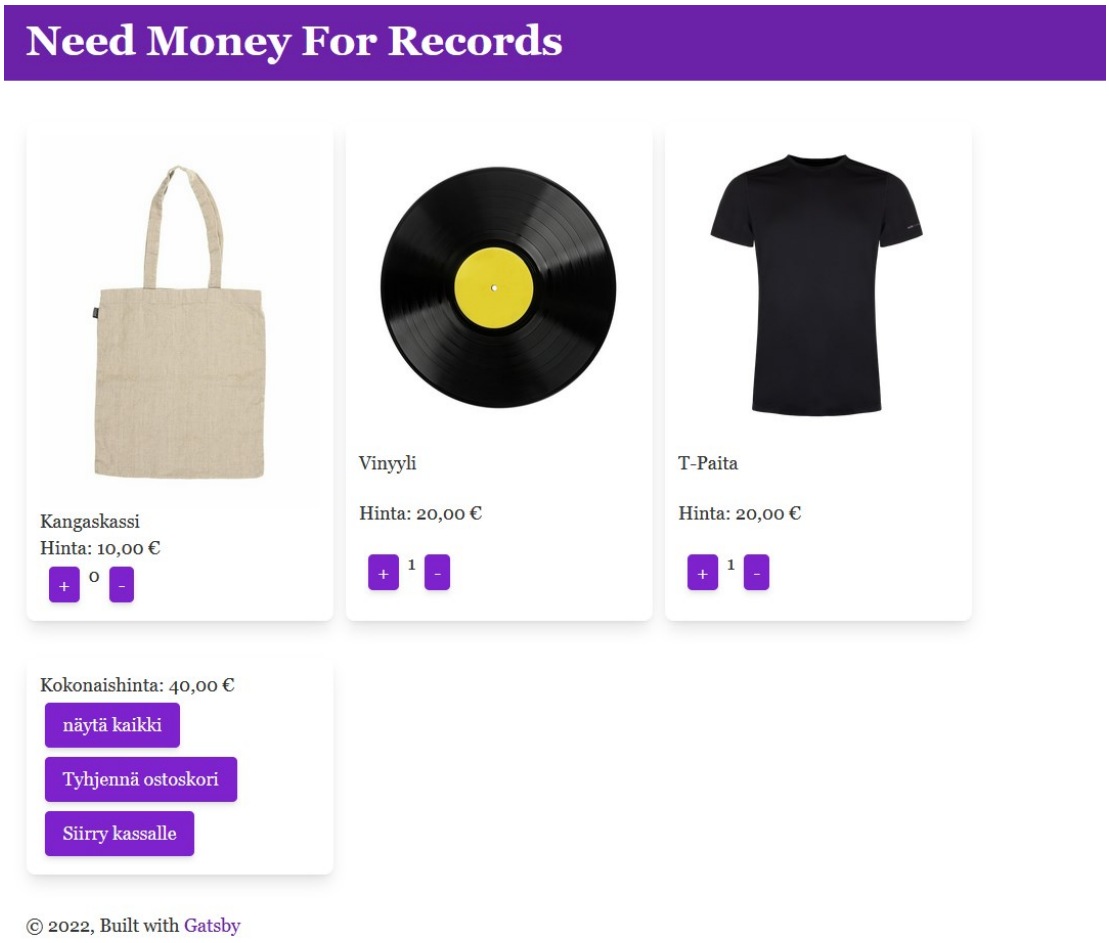
Kuva 15. Ostoskorin yksinkertainen ulkoasu.

Jos ostoskorissa painaa ”näytä kaikki”, niin sieltä tulee näkyviin tuotteiden nimi, hinta ja määrä (kuva 15).



Kuva 16. Ostoskorin tuotekomponentin tämänhetkinen ulkoasu.

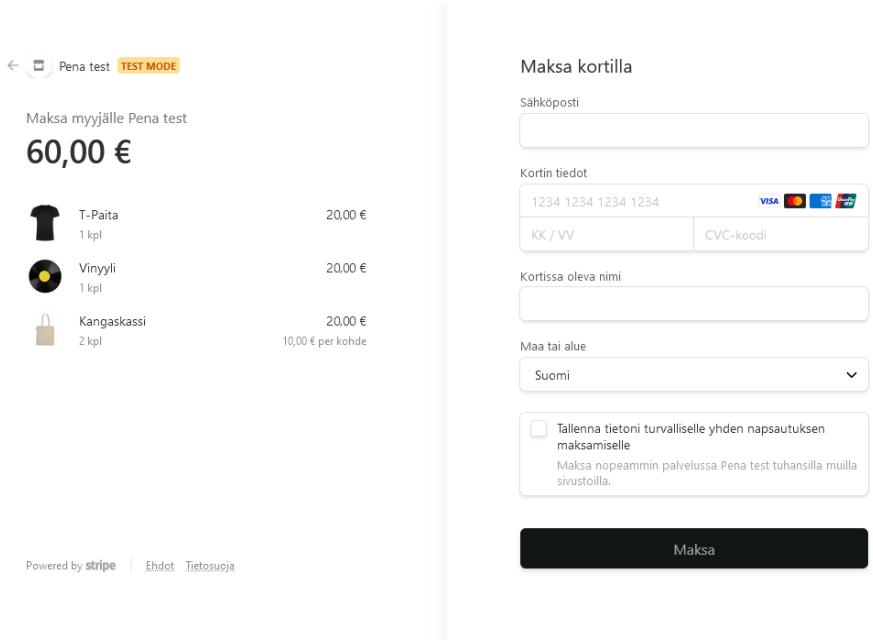
Kaikkien komponenttien rakentamisen jälkeen ulkoasu alkaa olemaan kasassa (kuva 16).



Kuva 17. Verkkokaupan ulkoasu.

3.5 Kassa

Stripen palvelu tarjoilee tarvittavan maksuominaisuuden. Omalta verkkosivulta käyttäjä ohjataan siis Stripen sivulle, jossa hän voi tehdä maksusuorituksen ja näkee vielä kerran, mitä on tilaamassa (kuva 18).



Kuva 18. Stripen kassa.

Tältä sivulta poistuessa käyttäjä lähetetään kehittäjän määrittelemälle onnistumis- taikka peruutussivulle [11].

4 Yhteenveto

Insinöörityön tarkoituksena oli luoda verkkokauppa, jolla näkee, tuoko se bändeille lisämyyntiä. Samalla perehdyin Gatsbyn, Stripen ja staattisten nettisivujen perusteisiin. Olen aikaisemmin kokeillut Reactin ja Webpackin kanssa aika paljon asioita, joten Gatsbyn kanssa asiat sujuivat erittäin sujuvasti. Stripen kanssa aikaa kului vähän enemmän, mutta siinä on helppo testailla kaikkia Stripen-toimintoja, joten kaikki tarvittavat toiminnot saatiin valjastettua. Insinöörityössä päästiin tavoitteisiin ja tarpeen vaatiessa verkkokauppaan olisi helppoa lisätä lisäominaisuuksia, kuten analyyttiset palvelut.

Stripen hintapolitiikka on ottaa pieni siivu jokaisesta tulleesta maksusta ilman mitään kuukausimaksua, joten sen käyttäminen pienissä projekteissa on erittäin suotavaa. Netlifyn osalta pienet projektit tuppaavat menemään aina ilmaiseen

aloitteluluokkaan, joten siitäkään ei ihan hirveästi kannata murehtia. Loppupeleissä tällaisessa pienessä projektissa ainoaksi jatkuvaksi maksuksi tulisi jäämään yleensä 12 € / vuosi oleva domain-nimi.

Stripen tarvittavien toimintojen opettaminen verkkokaupan loppukäyttäjälle on erittäin helppoa, koska Stripen käyttöliittymä on erittäin selkeä. Kehitysprosessi sujui loppupeleissä yllättävän hyvin johtuen hyvin ylläpidettävistä dokumentaatioista jokaisessa käytetyssä kirjastossa tai palvelussa. React-tiedot menivät yksi yhteen Gatsbyn kanssa. Tailwind on eri tavalla kirjoitettua CSS:ää. Netlifyhin koodin siirto ja CI/CD-putken virittäminen oli helppoa ja siihen sai vaivattomasti Stripen myös kiinni.

Loppupeleissä sivusto hoitaa, mitä varten se on tehtykin, mutta siinä voisi parantaa käyttäjäystävällisyyttä, siihen voisi lisätä analytiikan seurantasovelluksia tai vaikka testejä. Tässä ei lähdetty hakemaan täydellistä verkkokauppaa vaan kokeilumielessä testisivua, jonka tämä ajaa hyvin, vaikka se menisi roskiin tai jatkokehitykseen myöhemmin.

Lähteet

1 Jamstack-arkkitehtuuri. 2022. Verkkoaineisto. Knowit. <https://www.knowit.fi/palvelut/experience/web-ja-e-commerce/digitaaliset-palvelut-ja-alustat/jamstack/>. Luettu 26.10.2022.

2 Welcome to the Jamstack. 2022. Verkkoaineisto. Netlify. <https://www.netlify.com/jamstack/>. Luettu 26.10.2022.

3 What is CI/CD. 2022. Verkkoaineisto. Redhat. <https://www.redhat.com/en/topics/devops/what-is-ci-cd>. Luettu 4.11.2022.

4 What is Gatsby? 2022. Verkkoaineisto. Sean Ryan. <https://www.mparticle.com/blog/what-is-gatsby/>. Luettu 5.10.2022.

5 Concepts. 2022. Verkkoaineisto. Webpack. <https://webpack.js.org/concepts/>. Luettu 1.11.2022.

6 What is webpack? 2022. Verkkoaineisto. Survivejs. <https://survivejs.com/webpack/what-is-webpack/>. Luettu 1.11.2022.

7 Resolvers. 2022. Verkkoaineisto. ApollographQL. <https://www.apollographql.com/docs/apollo-server/data/resolvers/>. Luettu 9.11.2022.

8 Building online stores. 2022. Verkkoaineisto. Jordan Glover. <https://www.websitebuilderexpert.com/building-online-stores/>. Luettu 5.10.2022.

9 Tailwind CSS. 2022. Verkkoaineisto. Wikipedia. < https://en.wikipedia.org/wiki/Tailwind_CSS>. Luettu 5.11.2022.

10 Quick start. 2022. Verkkoaineisto. Gatsby. <https://www.gatsbyjs.com/docs/quick-start/>. Luettu 5.10.2022.

11 How checkout works. 2022. Verkkoaineisto. Stripe. <https://stripe.com/docs/payments/checkout/how-checkout-works>. Luettu 10.10.2022.