Blogin kehittäminen MERN pinolla

Susanna Hämäläinen

2022 Laurea

Laurea-ammattikorkeakoulu

Blogin kehittäminen MERN pinolla

Susanna Hämäläinen
Tietojenkäsittely
Opinnäytetyö
Huhtikuu, 2022

Laurea-ammattikorkeakoulu **Tiivistelmä**
Tietojenkäsittely
Trademoni(AMK)

Susanna Hämäläinen
Blogin kehittäminen MERN pinolla
Vuosi 2022 Sivumäärä 54

Tämän opinnäytetyön tarkoituksena oli kehittää sovelluskehitystaitoja web-sovelluksen suunnitelusta sekä sen kehittämisestä ja toimia näyttönä tästä osaamisesta. Sovelluksen kehittämiseen käytettiin MERN -pinoa, jolla luotiin sekä käyttöliittymä että palvelin.

Tavoitteena oli kehittää blogi, joka on lukijan näkökulmasta mukava käyttää. Samalla kehitettiin blogin ylläpitäjälle hallintajärjestelmä, jonka kautta blogin ylläpito, blogikirjoituksien tekeminen, muokkaaminen ja poistaminen on helppoa.

Opinnäytetyö toteutettiin kehittämistyönä ja tietoperustana toimi monenlaiset aiheeseen liittyvät e-kirjat ja ohjelmistotalojen koti- ja blogisivut. Kehittämistyössä tietoperustaa käytettiin sovelluskehityksen prosessin hahmottamisessa.

Kehittämistyön tuloksena syntyi toimiva blogin käyttöliittymä ja sen hallintajärjestelmä. Blogia voidaan tämän kehittämistyön jälkeen jatkokehittää, kun se otetaan aktiiviseen käyttöön. Jatkokehittämiseen kuuluu kaikille mahdollisuus tehdä käyttäjä sivulle, jotta blogikirjoituksia voi kommentoida. Myös hallintajärjestelmän parantaminen on tärkeä osa jatkokehittämistä.

Asiasanat: mern, web -kehitys, hallintajärjestelmä

Laurea University of Applied Sciences **Abstract**
Degree Programme in Business Information Technology
Bachelor's Degree

Susanna Hämäläinen
Developing a blog with MERN stack
Year 2022 Pages 54

The purpose of this bachelor's Thesis was to improve web development skills, from designing a website to developing it and to function as a proof of these skills. MERN stack was used in this development project to develop the frontend and the backend.

The goal was to develop a blog which feels good to use from the user's point of view. At the same time the management system for the blog was also developed to handle creating, modifying and deleting blog posts.

The thesis was done as a development project. The development process was based on information gathered from e-books and the websites and blogs by software companies.

The result of the project was a working blog, as both the user's UI and the management system function well. The blog is at the point where it is easy to build on and make it even better. Some new features could be that everyone could make a user to the blog and can comment on blog posts. The management system can also be improved with minor features and details.

Keywords: mern, web development, management system

Sisällys

1 Johdanto

Tämä kehittämistyö on osa tietojenkäsittelyn tradenomitutkintoa ja tarkoituksena on tutustua ja kehittää MERN -pinon avulla blogisivu ja sen hallintajärjestelmä. Kehittämistyön tarkoituksena on myös oman osaamisen kehittäminen ja toimia näyttönä osaamisesta ja kiinnostuksesta aihetta kohtaan. Kehittämistyö on tehty ilman toimeksiantajaa.

Ensimmäisenä tarkennetaan, mitä tämän kehittämistyön aikana kehitetään ja mitä ei. Tämän jälkeen käydään yleisesti läpi, mitä web -sovelluskehitys on ja sen vaiheita. Tähän sisältyy web -sovelluksen käyttöliittymän suunnittelu ja, miten MERN -pinolla kehitetään web -sovellus.

Tietoperustan jälkeen aloitetaan blogin kehittäminen. Ensimmäiseksi suunnitellaan käyttöliittymän ja hallintajärjestelmän ulkonäkö. Mobiilin käyttöliittymästä tehdään prototyyppi, jota testataan. Kun suunnittelut on tehty, aloitetaan kehittämään sovelluksen palvelin. Palvelimen jälkeen kehitetään blogin käyttöliittymä käyttäen Reactia.

Kehittämistyön lopuksi, blogia testattiin ja se julkaistiin. Tämän lisäksi käydään läpi, miten blogia voisi parantaa ja, mitä täytyy ottaa huomioon, kun blogikirjoituksia julkaistaan, jotta ne pysyvät eettisesti oikeina.

2 Työn lähtökohdat

Työn lähtökohtana toimii kiinnostus full stack -kehittämiseen. Kiinnostus sai alkunsa MERN -kurssin myötä, jonka jälkeen on ollut halu kehittää omia sovelluksia alusta loppuun. Sovelluksien kehittäminen kasvattaa omia taitoja, ja tulee auttamaan tulevaisuudessa näyttönä osaamisestani. Vaikka ammatissaan hoitaisi vain frontendin tai backendin, on hyvä tietää mitä kokonaisuudessa sovelluskehityksessä tapahtuu.

2.1 Työn lähtökohdat, tarkoitus ja tavoitteet

Sovellus- ja nettisivukehitys ovat kiinnostaneet minua jo jonkin aikaa, joten sen tekeminen opinnäytetyönä oli jo varsin ajoissa päätetty. Työn tarkoituksena on olla mahdollisimman monipuolinen, mutta samalla se ei saa olla liian laaja, jotta jokaisen vaiheen saa tehtyä mahdollisimman hyvin.

Aiheeksi valitsin blogin tekemisen alusta loppuun. Aihe syntyi karsimalla laajempia projekteja pienemmäksi, säilyttäen suuren osan tärkeimmistä toiminnoista. Blogin tekeminen on myös sellainen projekti, jossa tulee käytettyä erilaisia toimintoja, joita tulen tarvitsemaan myös työelämässä.

Kehittämistyön tavoitteena on kehittää yksinkertainen ja siisti blogi, ottaen huomioon parhaat mahdolliset käytänteet. Parhaita käytänteitä otetaan huomioon blogin ulkonäöstä koodin siisteyteen.

2.2 Työn kuvaus ja rajaus

Kehitettävä projekti on blogi, jossa voi lisätä, muokata ja poistaa blogikirjoituksia, jos käyttäjällä on valta siihen. Käyttäjä voi myös luoda ja poistaa uutisia, jotka näkyvät omalla sivulla. Tämän työn aikana vain minulle luodaan käyttö- ja pääsyoikeus hallintajärjestelmään. Blogi luodaan alusta asti, sille tehdään oma tietokanta, johon tallennetaan blogikirjoitukset,

uutiset ja käyttäjä. Blogin ulkonäkö suunnitellaan ottaen huomioon kohderyhmä ja erilaisia UI- ja UX-perusteita, jotta käyttäjäkokemus olisi mahdollisimman hyvä. Blogiin ei luoda virallisia blogikirjoituksia.

Blogikirjoituksille kehitän oman hallintajärjestelmän, jossa blogikirjoituksen voi luoda, muokata tai poistaa. Blogikirjoituksiin lisään mahdollisesti ominaisuuden lisätä kuvia tai videopätkiä tai videoupotuksia keskelle eri tekstikappaleiden väliin. Blogikirjoitukset ja uutiset kirjoitetaan käyttäen markdownia.

Kehittämistyössä keskitytään sovelluksen käyttäjäkokemukseen ja sen ulkonäköön. Työssä pidetään huoli siitä, että ainakin käyttäjällä on miellyttävä olo käyttää sovellusta, vaikka se hallintapuolelta saattaakin jäädä käytöltään haastavammaksi.

2.3 Keskeiset käsitteet

Backend	Websovelluksen osa, joka ei näy käyttäjälle. Toimii käyttöliittymän ja tietokannan välillä ja hoitaa käytäjän lähettämiä pyyntöjä.
Frontend	Käyttöliittymä, joka lähettää kutsuja backendiin.
Full stack	Sisältää backendin ja frontendin ja on tapa, jossa kehitetään sovelluksen molemmat osat yhtenä projektina.
GitHub	Graafinen käyttöliittymä git:in käyttöä varten.
MERN -pino	JavaScript pohjainen full stack -pino, joka koostuu tietokantajärjestelmästä, viitekehyksestä, ympäristöstä ja kirjastosta.
React	JavaScript -kirjasto ja osa MERN -pinoa. Tämän avulla kehitetään käyttöliittymä.
Node	Ympäristö, joka mahdollistaa JavaScriptin ajamista backendissä.
Express	Viitekehys, jota käytetään Node:ssa. Auttaa mobiili- ja websovelluksien kehittämisessä.
MongoDb	Dokumenttipohjainen oliotietokanta, joka kuuluu MERN -pinoon.
Token	Valtuutuskoodi, jolla saadaan väliaikaisesti käyttöliittymässä valtuudet ja saadaan onnistunut kirjautuminen palveluun tai käyttöliittymään.
Komentorivi	Ohjelma, jolla voidaan antaa tietokoneelle ohjeita erilaisilla komennoilla.

3 Web-sovellusten kehittäminen

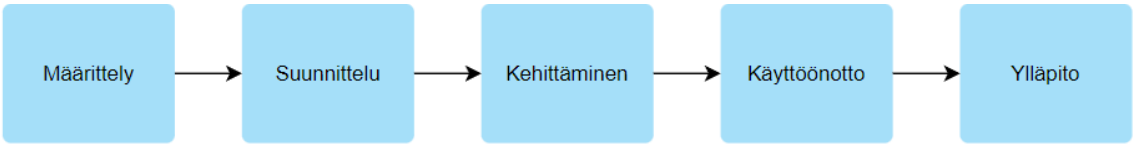
Web-sovellus on verkkosivu, joka pyörii selaimessa kuin tavallinen laitteelle ladattava sovellus (Redandblue 2021). Verkkosivu ei kuitenkaan ole sama asia kuin web-sovellus. Verkkosivun tarkoituksena on esittää dataa, web-sovelluksen tarkoitus on datan esittämisen lisäksi myös erilaiset interaktiiviset toiminnot, kuten haku -toiminto. Usein tavalliselle verkkosivulle voidaan lisätä myöhemmin toimintoja, mikä muuttaa sen verkkosivusta web-sovellukseksi. (Hughes 2019.)

Webkehitys ei ole pelkästään erilaisten verkkosivujen tai -sovelluksien kehittämistä, vaan siihen sisältyy myös näiden ylläpitäminen (Letendart 2018). Webkehitys viittaa verkkosivun tai

-sovelluksen ohjelmointiin, eikä sinänsä sisällä sovelluksen suunnittelu prosessia. Sovelluksen suunnittelu on kuitenkin tärkeä osa verkkosivun tai -sovelluksen kokonaisuutta, ja on usein prosessi, joka tehdään huolellisesti ennen kuin verkkosivun tai -sovelluksen ohjelmointi aloitetaan. (Bradford 2021.)

Verkkosivuja on monia erilaisia, ja niillä voi olla eri käyttötarkoituksia. Yksi yleisimmistä käyttötarkoituksista on yrityksen esille tuonti verkkoon, eli yritykselle tehdään verkkosivut. Tällöin mahdollinen asiakas voi tutustua yritykseen omaan tahtiin ja saa samalla tietynlaisen mielikuvan yrityksestä. Voidaan sanoa, että verkkosivut toimivat yrityksen käyntikorttina. (Yrityksen perustaminen.net 2021.)

Web-sovelluksen tai -sivun kehittämiselle voidaan määrittää viisi eri vaihdetta; määrittely, suunnittelu, kehittäminen, käyttöönotto ja ylläpito (kuvio 1). Nämä viisi määritettä luovat yhdessä prosessin, jota kronologisesti seuraamalla pystyy kehittämään kattavan web-sovelluksen tai -sivun, jos kukin osa on tehty kunnolla. (Lopuck 2012, luku 2.)



Kuvio 1: Web-sovelluksen kehittämisen viisi vaihetta(mukaillen Lopuck 2012)

Sovelluskehitystä tullaan tulevaisuudessakin tarvitsemaan, sillä web-sovelluksia joudutaan uusimaan välillä, jotta erilaiset yritykset pysyvät mukana maailman digitalisaatiossa (Pixels 2021). Websovelluksia voidaan kutsua myös progressiivisiksi websovelluksiksi, jos niiden tarkoitus on poistaa natiivien sovelluksien tarve. Natiivi sovellus on tietylle laitteelle ja käyttöjärjestelmälle kehitettävä ja ladattava sovellus. Progressiivinen websovelluksien kehittäminen tuo käyttäjälle natiivin sovelluksen tunteen, mutta se kuitenkin pyörii selaimessa. Progressiivisien websovelluksien kehittäminen yleistyy, sillä niiden kehittäminen on halvempaa kuin natiivien sovelluksien kehittäminen ja ne vaativat vähemmän muistia ja prosessorin käyttöä käyttäjän laitteella. (Ashutosh 2021.)

3.1 Käyttöliittymäsuunnittelu

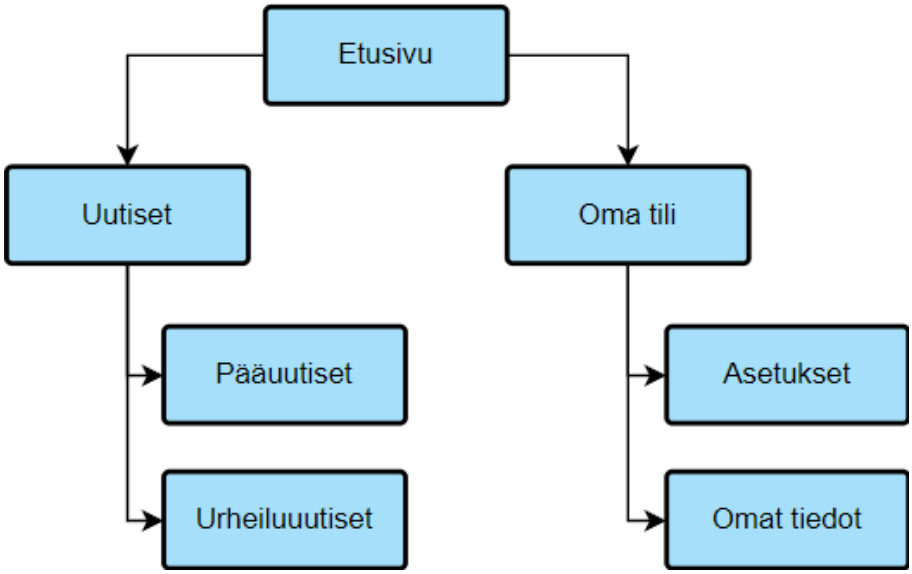
Käyttöliittymäsuunnittelun tarkoituksena on suunnitella yhtenäinen kokonaisuus kaikista sivun erillisistä alasivuista. Suunnitteluvaiheessa täytyy pitää mielessä, minkälaisen kuvan sivun omistaja, eli yritys tai yksityishenkilö haluaa itsestään antaa, ja miten käyttäjän kuuluu käyttäytyä sivulla käydessään. (Lopuck 2012, luku 1.)

Ennen kuin käyttöliittymää aletaan suunnittelemaan konkreettisesti, täytyy ensin määritellä tavoitteet ja vaatimukset. Tavoitteet ja vaatimukset auttavat antamaan kuvan web-sivun ulkonäöstä ja toiminnoista. Tavoitteet ja vaatimukset täytyy myös laittaa tärkeysjärjestykseen. (Lopuck 2012, luku 2.)

Kun kaikki tavoitteet ja vaatimukset ovat kasassa, myös niiden tärkeydet, on aika tehdä sivustokartta nettisivuista. Lopuckin (2012, luku 1.) mukaan, suunnittelu -vaihe on kuin kirjoittaisi kokoamisohjeet. Ilman oikeata suunnittelua ”tiimillä ei ole yhtenäistä suuntaa”.

Sivustokarttaan kootaan kaikki yhden nettisivun eri alasivut, miten ne linkittyvät toisiinsa ja niiden hierarkia. Sen voidaan ajatella olevan näkymä verkkosivuista lintuperspektiivistä. (Lopuck, luku 1.) Sivustokarttaa voidaan käyttää hakukoneoptimoinnissa. Siihen laitetaan myös tietoja sivulta löydettävistä videoista, kuvista ja artikkeleista, jotta hakukone osaa etsiä

myös näistä avainsanoja (Google search central 2021.) Yhdellä verkkosivulla voi olla useampi sivustokartta, sivun eri osista. Etusivulla voi olla oma sivustokartta (kuvio 2). Esimerkiksi blogissa blogikirjoituksilla voi olla oma sivustokartta, kuten myös sivun kuvilla. (Shinobi 2021.)



Kuvio 2: Esimerkki etusivun sivustokartasta

Kun sivustokartta on valmis, ja siinä on otettu huomioon eri sivut, voidaan ryhtyä tekemään rautalankamallia. Rautalankamalli keskittyy jokaiseen erilliseen kohtaan sivukartassa, ja esittää sen graafisesti, eli tässä vaiheessa on ajateltu, miltä sivun tulee näyttää. Rautalankamallissa keskitytään, mitä ja mihin kohtaa sivua mikäkin asia tulee, eli voidaan käyttää esimerkiksi erikokoisia laatikoita kuvastamaan elementin paikan sivulla. (Lopuck 2012, luku 2.)

Rautalankamallin jälkeen voidaan aloittaa suunnittelemaan, miltä nettisivu ja sen alasivut tulevat näyttämään, eli minkälaisen tunteen ne haluavat tai pitäisivät välittää käyttäjälle. Tähän vaiheeseen kuuluu eri värimaailmat, erilaiset kuvat ja niiden asetelmat. (Lopuck 2012, luku 2.)

Prototyyppi on suunnitteluista tehty malli, jonka avulla voidaan kokeilla, mitkä ominaisuudet voivat toimia milläkin tavalla. Prototyypissä on helppo muuttaa asioita, kun huomaa ettei tietty ominaisuus toimi kuten haluttaisiin. (Interaction design foundation 2020.) Prototyypeistä on paljon hyötyä, olipa kyseessä pieni tai iso projekti. Sen avulla säästää aikaa, rahaa ja sovellusta voidaan testata yhtenäisenä kokonaisuutena, vaikka kyseessä olisi eri laitteita. Paras prototyyppi on sellainen, jossa on lopullinen sisältö mahdollisen tarkasti mietitty. Jos tämä prototyyppi hyväksytään, on kehittäjien helppo tehdä sovellus suunnitelmien mukaan. (Coleman & Goodwin 2017, 3.)

Prototyyppejä on erilaisia. Niitä voi tehdä pelkällä paperille hahmottamalla tai digitaalisesti, usein käyttäen erilaisia tarjolla olevia sovelluksia. Myös rautalankamallia voidaan käyttää prototyyppinä. (Coleman & Goodwin 2017, 1–2.)

Käyttäjätestauksessa rautalankamalli annetaan testattavaksi, jotta virheitä voi korjata jo tässä vaiheessa. Käyttäjätestausta voidaan tehdä monessa eri vaiheessa, mutta prototyyppi vaiheessa se on kannattavaa, sillä silloin voi tutkia, miten käyttäjä oikeasti käyttäisi websovellusta. (Lopuck 2012, luku 1.) Voidaan sanoa, että rautalankamallista tehtyä prototyyppiä käytetään iterointikierroksilla. Iterointi tarkoittaa työvaiheen toistamista. Tässä tapauksessa prototyyppiä testataan uudelleen, kun edellisen iterointikierroksen tulokset on otettu huomioon ja tehty mahdollisia muutoksia.

Responsiivisessä sivussa sivun eri osat ja ulkonäkö muuttuvat sitä mukaan, kun näytön koko muuttuu (UXPin 2021a). Nämä muutokset voivat olla pieniä tai isoja, riippuen laitteesta. Muutos voi olla jopa niin pieni kuin parin pikselin muutos, usein leveyssuunnassa. (Graham 2015.) Kun sivua suunnitellaan responsiiviseksi, täytyy suunnitteluvaiheen alussa ottaa huomioon eri laitteet ja niiden näyttöjen koot. Sivun osat voivat liikkua valuvasti, jonka takia sivun eri osilla on oma hierarkiansa. Sivun hierarkialla tarkoitetaan, miten sivun eri elementit kuuluvat suurempiin kokonaisuuksiin, jotka taas kuuluvat vielä suurempaan kokonaisuuteen. (UXPin 2021a.)

Adaptiivisessa suunnittelussa suunnitellaan sivut eri laitteille, ei sinänsä laitteen koolle. Adaptiivinen sivu jaetaan pysäytyspisteisiin. Pysäytyspisteet ovat usein laitteiden mukaan. Yksi laite on yksi pysäytyspiste, mutta niitä voi olla useampikin. Adaptiivisessa suunnittelussa suunnitellaan jokaiselle laitetyypille oma pohja, joka otetaan käyttöön, kun sivu tunnistaa mikä laite on käytössä. (Graham 2015.)

Responsiivisen sivun tekeminen ja ylläpitäminen on helpompaa kuin adaptiivisen. Adaptiivisella sivulla voidaan tehdä monimutkaisempia ja tarkemmin kohdelaitteelle konfiguroituja sovelluksia. Adaptiivinen sivu vastaa nopeammin kuin responsiivinen sivu. Adaptiivisen sivun kehittäminen on vaikeampaa ja vie enemmän aikaa, mutta siinä voidaan ottaa paremmin huomioon käyttäjäkokemus. (UXPin 2021a.)

Mobile first -ajattelumallissa suunnitellaan sovellus käytettäväksi erilaisille laitteille, mobiililaitteesta tavalliseen työpöytäkoneeseen. Mobile first- ajattelutapa tuo lopputuotteelle arvoa. (Gonzales 2013, 19.) Mobile first -ajattelumallissa käyttöliittymä suunnitellaan ensin mobiililaitteelle, ja sen jälkeen vasta muille, sillä mobiilille on hankala suunnitella toimivia käyttöliittymiä. Tässä ajattelumallissa pidetään huoli siitä, että tärkeimmät ominaisuudet toimivat mobiililla, sillä siinä on rajattu näytön koko. (UXPin 2021b.)

3.2 Web-sovellusten kehittäminen MERN -pinolla

MERN-pino koostuu neljästä osasta, MongoDB, Express, React ja Node. Nämä neljä yhdessä muodostavat full stack-pinon, joka painottuu JavaScriptin käyttöön. MERN-pino on tarkoitettu websovelluksien kehittämiseen. (Hoque 2020, luku 1.)

Full-stack -pino on kokonaisuus, joka sisältää sekä serveripuolen että käyttöliittymäpuolen ohjelmoinnin. Tähän pinoon kuuluu myös tietokannan käyttö. Full stack-pinolla voidaan kehittää sovellus alusta loppuun. (StaffMill 2021.)

MERN -pinon valinta projektin kehittämiseen on vain yksi monesta vaihtoehdosta. Tämä pino kannattaa, jos haluaa käyttää yhtä pääohjelmointikieltä, JavaScript, ja, jos tietää käyttävänsä JSON-dataa ja dynaamisia web-käyttöliittymää. (MongoDB 2021.) MERN-pinon eri osilla on erilaisia ominaisuuksia, jotka tekevät tästä pinosta yksinkertaisen, mutta vaikuttavan (Hoque 2018, 6).

MERN-pinoa on helppo oppia, kehittää ja laajentaa, sillä pinon jokainen osa perustuu JavaScriptiin. Pinoa käytetään laajasti toimialalla ja sillä on yhteisö, joka tukee ja kasvaa koko ajan. (Hoque 2020, 17) MERN -pino on avointa lähdekoodia. Avoin lähdekoodi tarkoittaa ohjelmistoa tai sen osaa, jonka lähdekoodi on saatavilla julkisesti. (Leppäniemi 2021.)

Kehittämisen parhaisiin käytänteisiin kuuluu selkeän koodin kirjoittaminen. Sanotaan, että koodin pitäisi olla niin kirjoitettu, että henkilö, joka sitä ei ole kirjoittanut, pystyy ymmärtämään, mitä koodin on tarkoitus tehdä. Koodin ymmärtämistä varten on hyvä lisätä kommentteja selittämään lyhyesti, esimerkiksi mitä funktion on tarkoitus tehdä. Myös muuttujien ja funktioiden nimeäminen ovat tärkeä osa kehittämisen parhaita käytänteitä. Niitä on helpompi käyttää, jos niiden nimistä selkeästi näkee niiden tarkoituksen.

3.2.1 Backendin kehittäminen

Backend on websovelluksen osa, joka pysyy niin sanotusti piilossa käyttäjältä. Tätä puolta kutsutaan myös sovelluksen serveripuoleksi. Se ottaa vastaan käyttäjän pyyntöjä, varastoi dataa ja hoitaa sen kulkua serverin ja käyttöliittymän välillä. (Codecademy 2021.) Esimerkkejä käyttäjän tekemistä pyynnöistä ovat haun käyttäminen ja käyttäjätunnuksien luominen. Molemmissa käyttäjä haluaa tehdä käyttöliittymässä jotain, joka vaatii serverin vastaamista, sillä haettu data ei ole käyttöliittymän käytettävissä koko ajan. (Lopuck 2021, luku 2.)

Tietokanta on osa backendiä, sillä siihen otetaan yhteys käyttäen backendin ohjelmointikieliä (Codecademy 2021). Tietokanta on rekisteri, jonka tarkoituksena ylläpitää dataa, ja antaa tätä dataa erilaista käyttöön varten (Foster & Godbole 2016, 3). Tietokantatyyppejä on erilaisia, relaatio- ja oliotietokanta. **Relaatiotietokannassa** data esitetään taulukkomuodossa. Tällainen tietokanta koostuu riveistä ja kolumneista ja jokaisella rivillä on oma ainutlaatuinen tuntomerkki usein esitettynä ID-numerona. Tätä ID:tä kutsutaan myös avaimeksi. Relaatiotietokanta on selkeä ja todella yleinen tietokantamalli, sillä se on ollut käytössä jo 70-luvulta asti. (Oracle 2021.) **Oliotietokannassa** säilytetään dataa omina olioina ja jokaisella oliolla on omia ominaisuuksia ja metodeja. Tällaisessa tietokannassa ei ole selkeää taulukkoa, johon data on syötetty. Oliotietokantaa käytetään usein olio-ohjelmointikielien kanssa. (MongoDB 2021.)

Node, Express ja MongoDB ovat osa MERN -pinon backendiä. **Node** kehitettiin 2009, jotta JavaScriptiä voidaan käyttää sovelluksen backendissä. Node mahdollistaa kevyiden ja skaalautuvien sovelluksien kehittämisen, sillä se on asynkroninen, eli sovelluksen eri osat eivät ole toisistaan riippuvaisia ja voivat ajaa sovelluksen eri osia, vaikka jokin osa epäonnistuu. Noden kautta voi ottaa käyttöön paketinhallintajärjestelmä *npm*:n, jonka avulla sovellukseen voi ladata erilaisia kirjastoja. Nämä kirjastot tekevät sovelluksen ohjelmoinnista helpompaa. (Hoque 2020, 13.)

Express on yksi Noden viitekehyksistä. Expressistä on apua mobiili- ja websovelluksien kehittämisessä, sillä se sisältää monia toimintoja, jotka eivät sekoita eikä vaikeuta tavallisen Noden käyttöä. (Express 2021.) Expressin mukana tulee väliohjelmistotoimintoja. Näillä toiminnoilla käsitellään pyyntöjä ja vastauksia, kuten API-pyyntöjä tietokantaan. Express voi toimia myös väliseinänä käyttöliittymän saaman tiedon ja serverillä olevan tiedon välillä. (Hoque 2020, 14–15.)

API on ohjelmointirajapinta, joka mahdollistaa tietojen haun erilaisilta sivuilta, jotka ovat antaneet luvan siihen. Tietojen hakuun käytetään API-pyyntöjä, jotka ovat erikseen määritelty. (Visma 2022.) API-pyyntöön tarvitaan osoite, johon pyyntö lähetetään ja tieto, minkälainen pyyntö lähetetään. On neljä pyyntöä, jotka löytyvät jossakin muodossa kaikista ohjelmointirajapinnoista, GET, POST, PUT ja DELETE. (Fitzgerald 2022.)

MongoDB on dokumenttipohjainen oliotietokanta, minkä takia sitä on helppo laajentaa tarpeen mukaan. MongoDB säilyttää dataa JSON-muodossa. (Hoque 2020, 15.) JSON on yksi monesta eri tyypistä, miten dataa voi varastoida tietokannassa. Se eroaa muista olemalla helposti luettavaa sellaisenaan. (JSON 2021.) Kuviossa 3 on esimerkki, miltä JSON -data näyttää. Siinä näkyy selkeästi, muuttujan nimi vihreällä ja sen arvo punaisella.

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [],
  "spouse": null
}
```

Kuvio 3: Data JSON -muodossa (Wikipedia 2022)

3.2.2 Frontendin kehittäminen

React on MERN -pinon osa, joka näkyy käyttäjälle. Reactia käytetään käyttöliittymien kehittämiseen. Se mahdollistaa yksittäisten komponenttien päivittämisen sitä mukaan, kun ne muuttuvat ilman, että koko sivua tarvitsee päivittää uudelleen. Yksittäisten komponenttien päivittäminen on tehokasta, ja vähentää ylimääräistä prosessorin käyttöä. Tämä antaa käyttäjälle paremman käyttäjäkokemuksen ja sovelluksen käyttö tuntuu kevyemmältä. (React 2021).

Reactissa sivut luodaan yhdistämällä eri komponentteja. Näitä komponentteja voidaan myös uudelleen käyttää muissa projekteissa. Reactin käyttäminen pakottaa ohjelmoijia ohjelmoimaan tavalla, joka helpottaa myös koodin testaamista ja sen laajentamista. (Hoque 2020, 16).

4 Käytetyt kehittämismenetelmät ja -työkalut

Ketteriä menetelmiä käytetään projektihallinnan välineenä. Ne mahdollistavan joustavuuden projektissa, eikä niissä seurata suoraviivaista suunnittelua. Ketterissä menetelmissä muutoksia on helppo tehdä, sillä ne ovat joustava ja muutosmyönteisiä menetelmiä. Tunnettuja ja suosittuja ketteriä menetelmiä ovat mm. Scrum, Lean ja Kanban. (Koulutus.fi 2021.) Tässä kehittämistyössä käytetään Scrummia ja Kanbania yhdessä.

Scrumissa isompi tiimi jaetaan pienempiin tiimeihin ja jokainen pieni tiimi hoitaa pienen osan isompaa kokonaisuutta. Scrumissa nämä pienet tiimit tekevät työnsä sprinteissä tai iteraatioissa. Sprintit kestävät usein kaksi viikkoa. Sprintin alussa asetetaan tavoite, joka on tarkoitus saavuttaa sprintin aikana. (Hietaniemi 2019.) Tässä kehittämistyössä käytetään vain tiettyjä ominaisuuksia Scrumista, sillä Scrum on tarkoitettu tiimille, mutta tämä

kehittämistyö tehdään yksin. Kehittämistyössä tehdään sprinttejä, joilla on tavoite. Sprintin loputtua katsotaan, päästiinkö tavoitteeseen ja arvioidaan tekeminen.

Kanban -työkalun on tarkoitus näyttää selkeästi, mitä asioita pitää tehdä, mitkä ovat työnalla ja mitkä valmiita. Kanbanissa tehtäviä voi luokitella eri tavalla, esimerkiksi antamalla tehtäville tärkeysasteen, jonka mukaan tehtävät suoritetaan. Tehtävät voidaan myös jakaa eri henkilöille, ja henkilölle voidaan antaa vain tietty määrä tehtäviä. Näin yhdelle henkilölle ei tule enemmän tekemistä kuin muille ja on helppo nähdä, kuinka paljon eri henkilöillä on tekemistä. (Hietaniemi 2020.) Kanban -työkalusta tullaan käyttämään tehtävien jaottelua eri prosessin vaiheisiin ja tehtäville annetaan tärkeysaste, jonka avulla tärkeimmät tehtävät tehdään ensin.

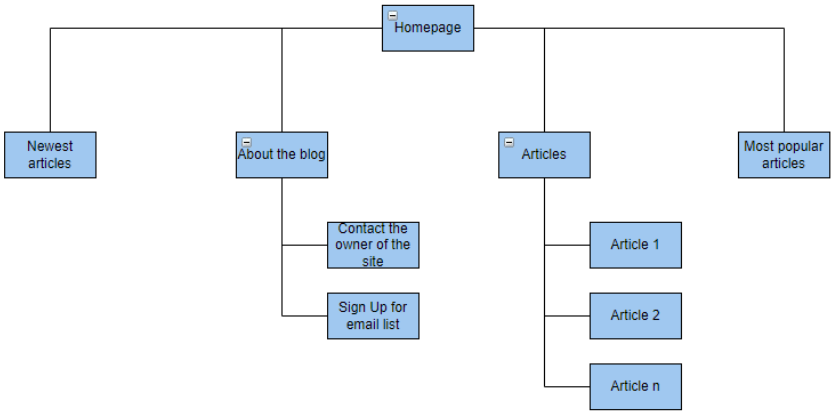
Jira on ohjelmistokehitystyökalu. Se sisältää erilaisia tauluja, kuten Kanban- ja Scrum -tauluja. Jiran avulla voidaan myös ajoittaa eri tehtävät ja seurata, kuinka hyvin tavoitteet on saavutettu. Jira antaa jokaiselle yksittäiselle tehtävälle oman avaimen. Tätä avainta käytetään, kun halutaan tehdä tätä tehtävää. Jiran saa yhdistettyä GitHub:iin, jota käytetään versionhallintaan ja ylipäätänsä koodin säilyttämiseen. (Atlassian 2021.) Tässä kehittämistyössä käytetään Jiran antamia työkaluja kehittämistyön prosessin seuraamiseen ja hallitsemiseen. Jiraa ei yhdistetä GitHubiin, vaikka koodia säilytetään ja ajetaan sitä kautta.

5 Kehittämistyön käyttöliittymän suunnittelu

Blogin aiheeksi olen valinnut todellisista rikoksista kertovan blogin. Koska aihe on maailmanlaajuinen, olen myös päättänyt, että blogin sisältö luodaan englanniksi. Aiheen takia, kohderyhmää ei voida rajata selkeästi, kuka tahansa voi olla kiinnostunut aiheesta. Tämän takia kohderyhmäksi olen yleisesti rajoittanut henkilöt, jotka arkena käyttävät selainta ja käyvät erilaisilla nettisivuilla.

Blogin tavoitteena on olla mahdollisimman helppokäyttöinen lukijalle. Lukijan täytyy pystyä siirtymään sivuilla sujuvasti erilaisilla laitteilla ja tekstiä pitää pystyä lukemaan helposti. Blogin hallitsijan tulee pystyä hallitsemaan blogin sisältöä helposti ja nopeasti.

Kuviossa 4 osoitetaan, miltä blogin sivukartta voisi näyttää. Se voi muuttua, mitä pidemmälle sivua kehitetään. Sivukarttaa tehdessä tulee mietittyä miten liikkuminen sivujen välillä voi tapahtua, ja mitä yleisiä linkkejä sivuilla voi olla. Kuvion 4 sivukartta ei sisällä kaikkia sivuja, joita kehitetyille sivuille tulee, sillä tässä vaiheessa sen tarkoitus on auttaa hahmottamaan kokonaisuutta minulle.



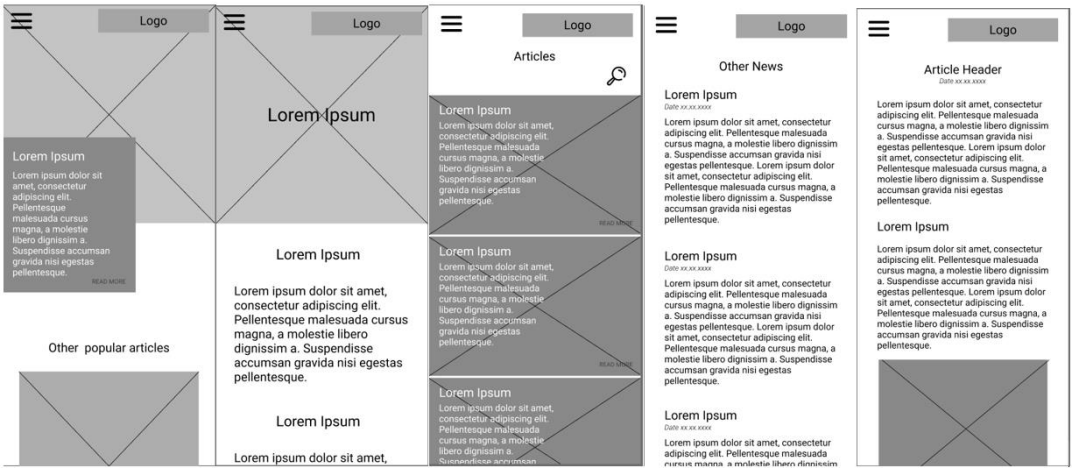
Kuvio 4: Sivukartta etusivusta nähden

Kehittämistyöstä tehdään seuraavaksi **rautalankamalli**, jonka avulla on helppo nähdä, mihin sivun eri elementit voi sijoittaa ja, miltä lopputulos suunnilleen näyttää. Rautalankamallin tekemiseen on käytettävissä erilaisia työkaluja. Nämä työkalut helpottavat sellaisen tekemistä ja muokkausta. Tässä kehittämistyössä on käytetty **Figma** -nimistä työkalua, sillä työkalulla voi tehdä myös yksinkertaisia prototyyppejä mobiililaitteelle. Suunnittelussa on otettu huomioon, että blogista tehdään responsiivinen.

Blogin suunnitteluvaiheessa on tutustuttu jo olemassa oleviin blogeihin. Näin voidaan tutustua siihen, miten muut käyttöliittymät ovat ratkaisseet erilaiset siirtymät sivujen välillä, ja millä tavalla on herätetty lukijan huomio eri sivun osiin. Kun muista samantapaisista blogeista ottaa mallia voidaan myös varmistaa, että käyttäjälle on tuttua, miten sivulla voi liikkua, eli liikkuminen sivulla on intuitiivista.

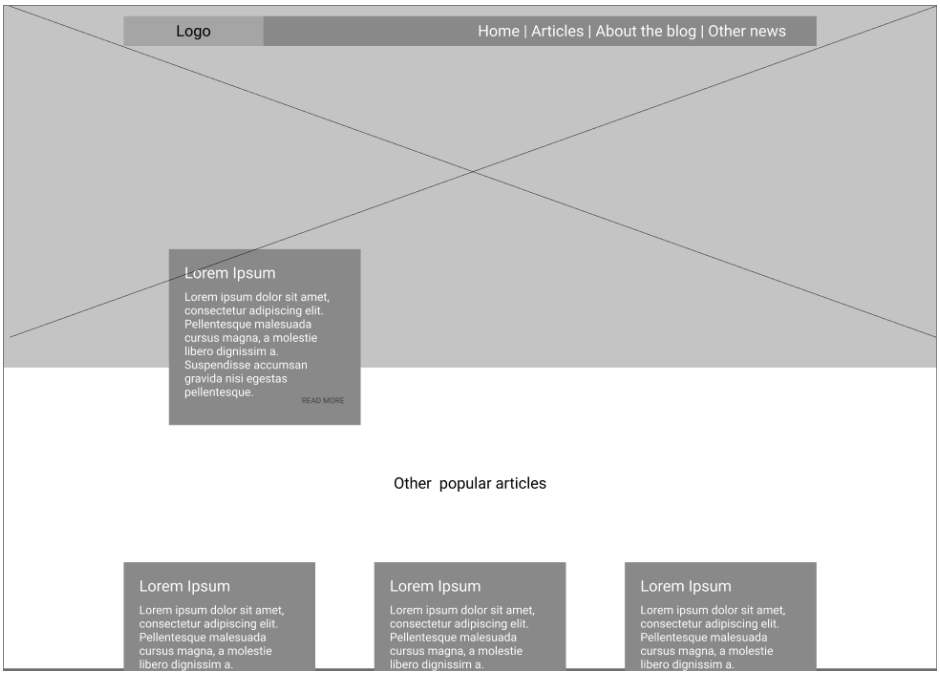
Kuviossa 5 näkyy, miltä blogin eri sivut tulevat suurin piirtein näyttämään mobiililaitteella. Mobiililaitteita on monen eri kokoisia, jonka takia jokaiselle eri koolle ei voida tai kannata suunnitella omaa ulkonäköä. Suunnitelman on tarkoitus antaa ohjeita, mihin eri elementit sijoitetaan ruudulla. Kuviosta 5 näkee selkeästi, että mobiililaitteella navigaatio tulee sivun vasempaan yläreunaan ja logo oikeeseen yläreunaa. Vaikka blogin suunnittelussa käytetään mobile first -ajattelumallia, on hyvä samalla pitää mielessä, miltä samat komponentit voivat näyttää suuremmalla näytöllä.

Mobiililaitteella on käytössä hampurilaisnavigaatio. Hampurilaisnavigaatio on erillinen navigaatio, joka avautuu painamalla tiettyä nappia tai kuviota. Kuviossa 5 hampurilaisnavigaatio on vasemmassa ylänurkassa ja siitä painamalla avautuu navigaatio, jossa on listattu kaikki sivut, jolle on mahdollista päästä.



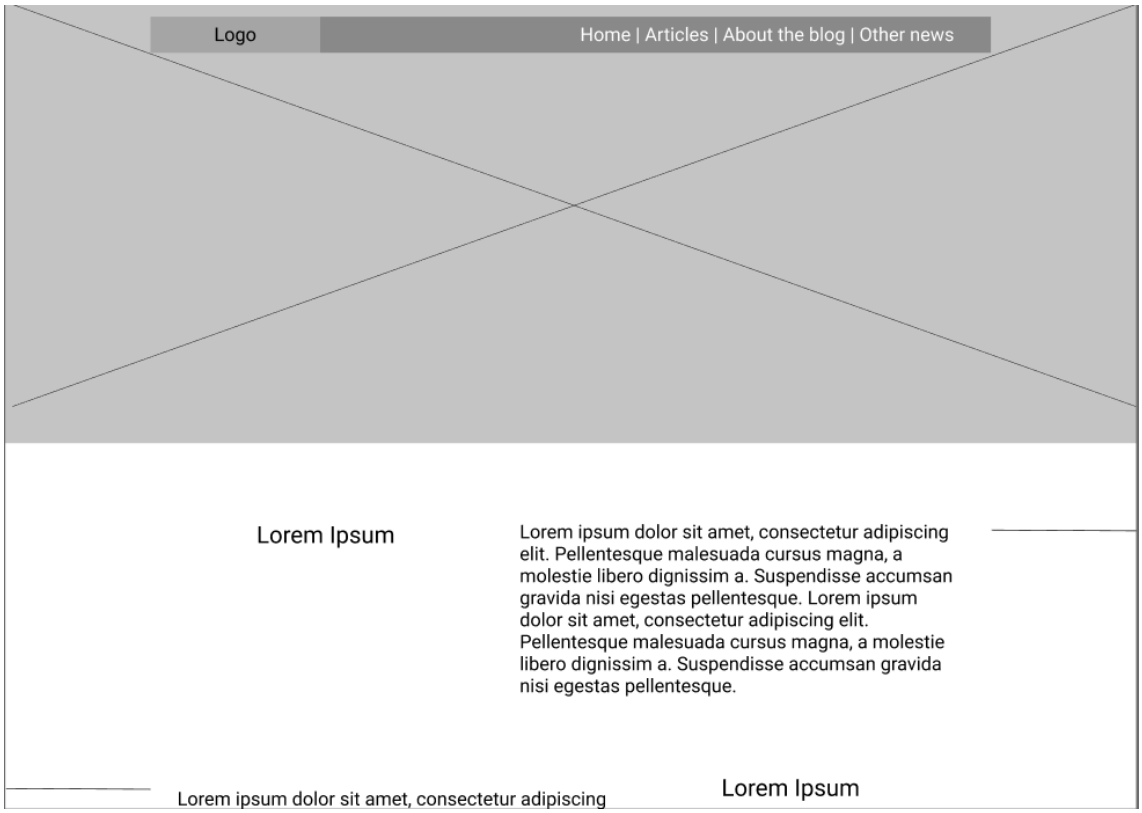
Kuvio 5: Mobiililaitteen rautalankamallit sovelluksen sivuista

Kuviossa 6 on käytetty pohjana mobiiliversion etusivun rautalankamallia. Siinä on otettu kaikki samat elementit ja kokonaisuudet ja sijoitettu uudelleen suuremmalle näytölle. Suuremmalla näytöllä on enemmän tilaa sivun eri elementeille, jonka takia navigoinnin ulkonäkö on toteutettu eri tavalla kuin kapealle mobiililaitteelle. Koska kuvion 6 näyttö on leveämpi, voidaan blogikirjoituksien kortit laittaa myös vierekkäin, toisin kuin kapeammalle näytölle, kuten mobiililaiteelle.



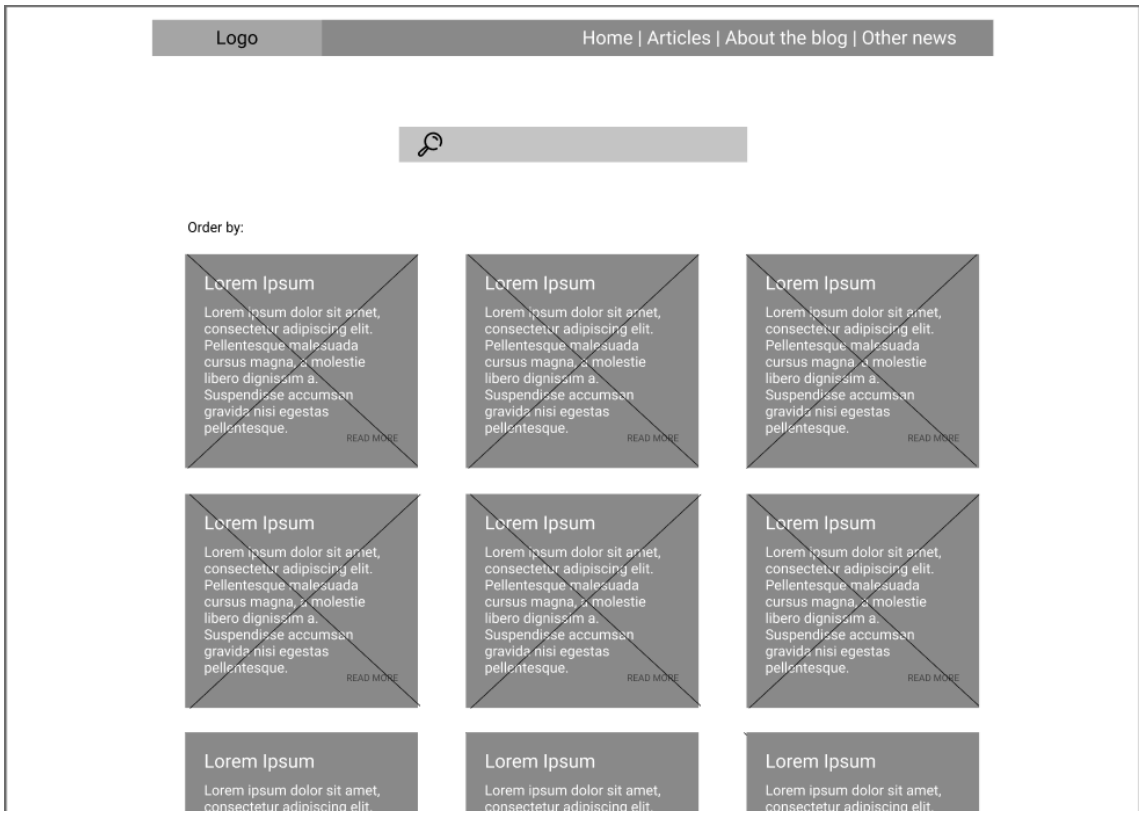
Kuvio 6: Rautalankamalli etusivusta isommalla näytöllä

Kuviossa 7 on rautalankamalli *about the blog* -sivulle. Sivun ylälaidassa tulee olemaan sivun banneri, jonka koko riippuu näytön koosta. Kuviossa 7 banneri on reunasta reunaan, mutta suuremmilla näytöillä tämä ei kannata, jotta lukijan huomio keskittyy sivun keskelle. Bannerin tarkoitus on kertoa käyttäjälle nopeasti, millä sivulla hän on. Sivu tulee olemaan myös tekstipainotteinen, jonka takia teksti on jaettu selkeästi eri osiin.



Kuvio 7: Rautalankamalli *about the blog* -sivusta

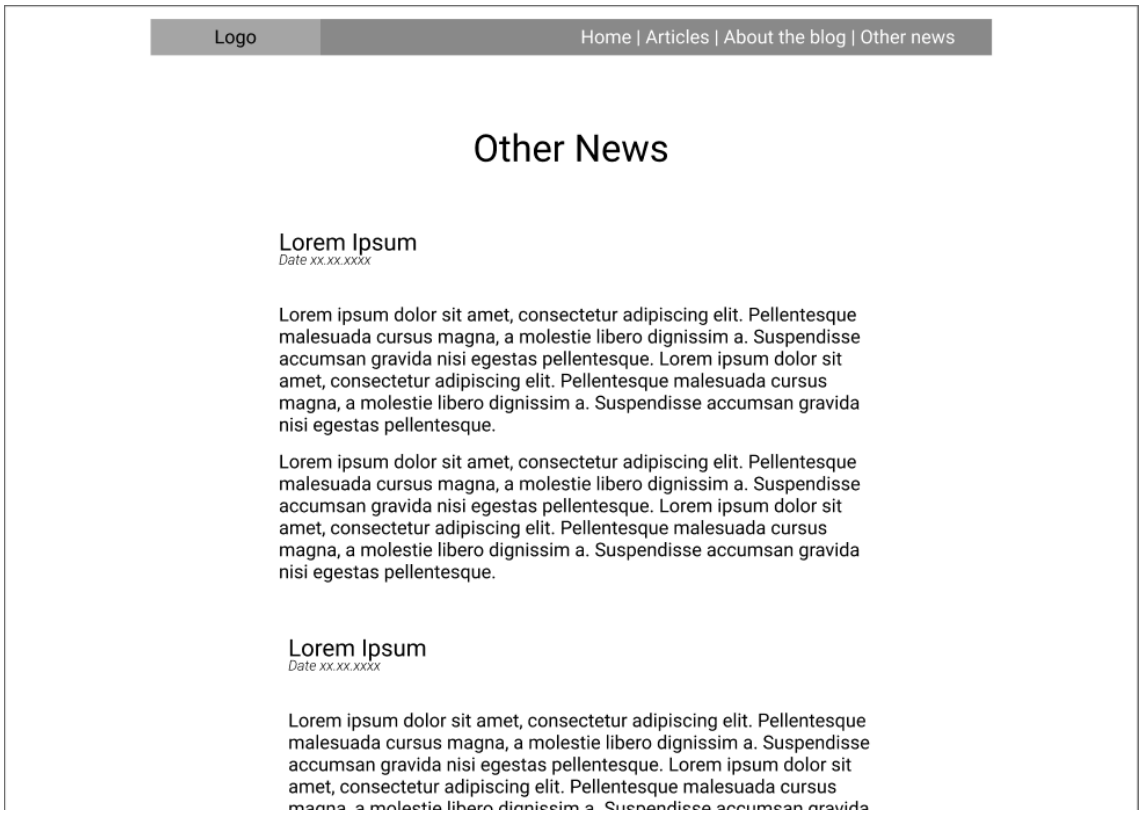
Kuviossa 8 on suunniteltu, miltä *articles* -sivu tulee näyttämään. Sivulla on myös hakukenttä, jonka avulla käyttäjä voi hakea blogikirjoituksien otsikoiden mukaan haluamansa blogikirjoituksen.



Kuvio 8: Rautalankamalli *articles* -sivusta

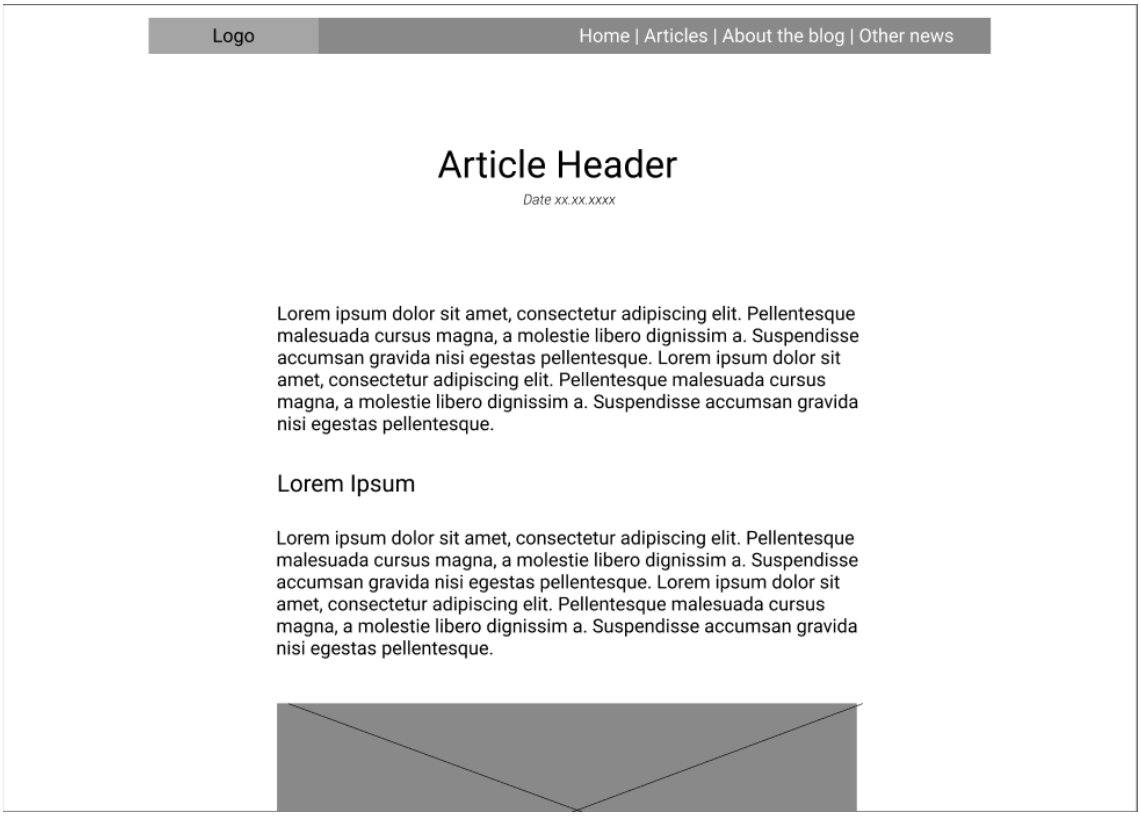
Isommalla näytöllä blogikirjoitukset listataan kolme blogikirjoitusta per rivi. Jokaisesta julkaisusta näkyy kuva, otsikko ja teksti, joka kertoo lyhyesti, mistä se kertoo. Mobiililaitteella blogikirjoitukset listataan allekkain, sillä yhden blogipostauksen kortin koko täytyy olla tarpeeksi iso, jotta käyttäjä saa selkeästi luettua otsikon ja tekstin, jotka siinä ovat.

Other news -sivun on tarkoituksena uutisoida ja jakaa tietoja käyttäjälle. Kuviossa 9 näkyy, että sivu tulee olemaan myös tekstipainotteinen, mutta uutisten on tarkoitus olla lyhyitä, joten otsikot jakavat tekstin selkeästi pienempiin osiin. Uutisissa on otsikko, päivämäärä ja itse uutinen. Koska uutisten ei ole tarkoitus olla pitkiä, on myös niiden leveys määrätty kapeammaksi, kuin muilla sivuilla oleva sisältö.



Kuvio 9: Rautalankamalli *other news* -sivusta

Kuviossa 10 on suunniteltu, miltä itse blogikirjoitus näyttää, kun sitä haluaa lukea. Suunnitelman on tarkoitus näyttää, mitä sivulle voi laittaa. Sivun yläreunassa on blogikirjoituksen otsikko ja päivämäärä, jolloin se on julkaistu. Blogikirjoituksiin voidaan laittaa eri tasoisia otsikoita, tekstiä ja kuvia, joissa on kuvateksti.



Kuvio 10: Rautalankamalli blogikirjoituksen sivusta, kun sen avaa luettavaksi

Seuraavaksi valitaan blogin värimaailma. Siihen vaikuttaa blogin aihe, sillä eri värit herättävät eri tunteita. Lyhyen tutkinnan jälkeen olen päättänyt valita liilan pääväriksi. Liila luo sekä rauhallisen että surullisen tunteen ja se kuuluu viileiden värien ryhmään. Viileät värit ylipäätänsä antavat tunteen rauhallisuudesta ja vakavuudesta.

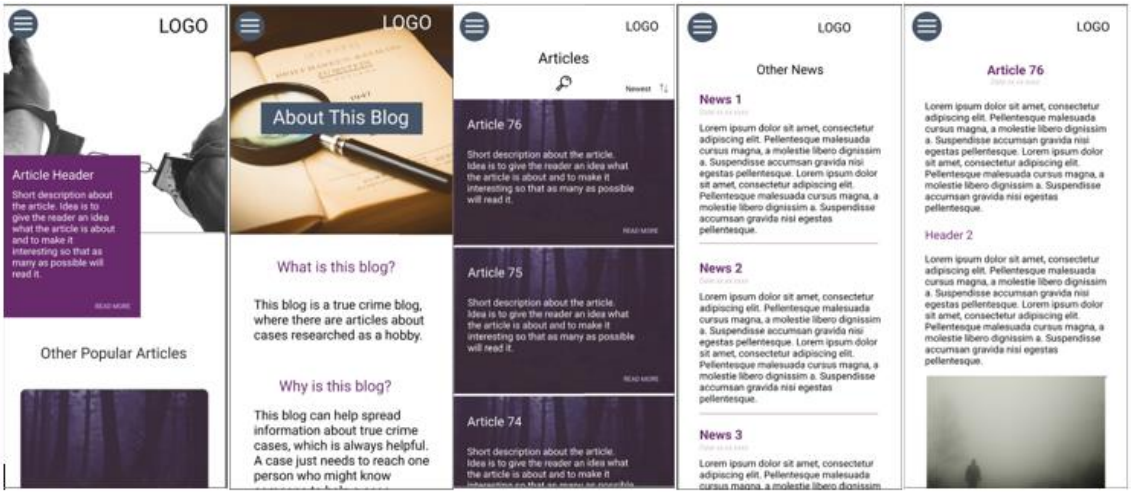
Värimaailman valittaessa täytyy ottaa myös huomioon värien tarpeeksi suuret kontrastit. Värejä verrataan neutraaliin harmaaseen laittamalla se kaikkien valittujen värien taustalle (kuvio 11). Neutraali harmaa tausta auttaa näkemään, onko väreissä tarpeeksi suuri kontrasti ja sopivatko ne myös yhteen.

Kuvio 11: Blogille suunniteltu värimaailma

Sivulle luodaan myös väliaikainen sisältö. Tässä vaiheessa väliaikainen sisältö tarkoittaa, että on tarkennettu, minkälaista tekstiä tai kuvia tulee sivulle ja mihin. Esimerkiksi sivulla *about the blog* on kerrottu tarkemmin, minkälaista tekstiä missäkin kappaleessa voi olla. Myös blogikirjoituksien kortteihin on laitettu tilapäiseksi sisällöksi ohje, mitä siinä voi kertoa blogikirjoituksesta.

Kuviossa 12 näkyy, miltä blogi tulee näyttämään mobiililaitteella värien ja kuvien kanssa. Kaikkia valittuja värejä on käytetty, toisia enemmän kuin muita. Värit on sijoitettu niin, että niiden välillä on mahdollisimman suuri kontrasti, vaalea tummalla taustalla ja tumma

vaalealla taustalla. Sivun eri elementteihin on myös sijoitettu väliaikainen sisältö, joka saattavat muuttua, mitä pidemmälle kehittämistyössä edetään.

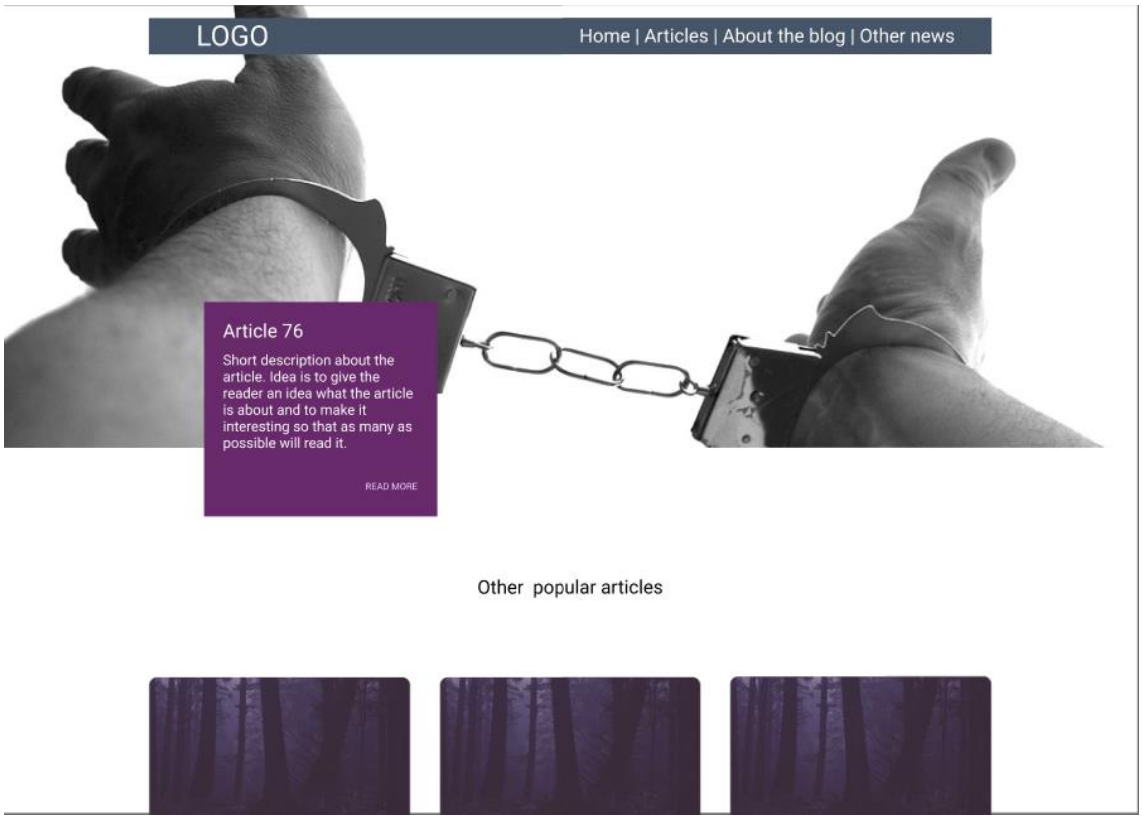


Kuvio 12: Mobiililaitteelle suunniteltu ulkonäkö väliaikaisella sisällöllä ja värimaailmalla

Kuviossa 13 on mobiililaitteelle tarkoitettu navigointi. Se avautuu painamalla vasemmasta yläkulmasta, ja sen tarkoitus on avautua sujuvasti animaation avulla kuvion 13 näköiseksi. Mobiililaitteen navigaatiossa pitää ottaa huomioon, että näyttöä käytetään todennäköisesti sormilla, eli erilaiset linkit eivät saa olla liian lähellä toisiaan ja klikattava alue täytyy olla tarpeeksi suuri.

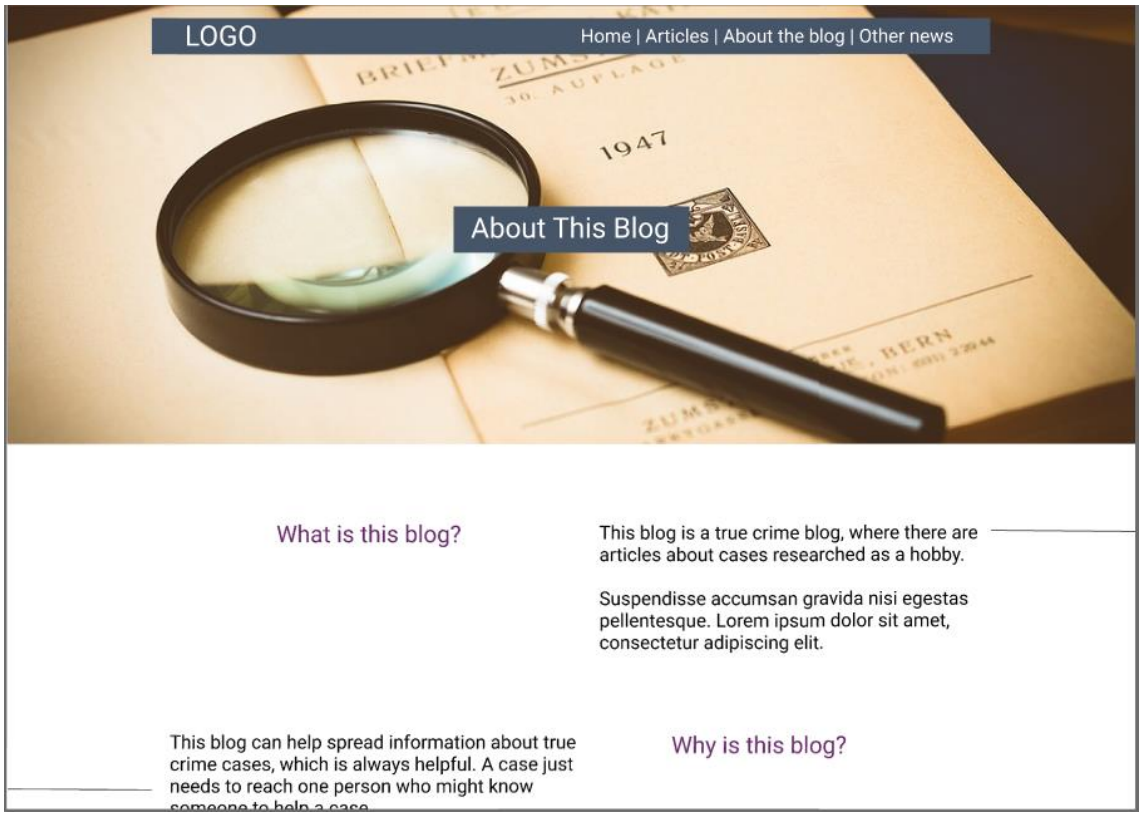
Kuvio 13: Mobiililaitteen navigaatio

Kuviossa 14 bannerin koko riippuu näytön koosta, mutta isommilla näytöillä koko sisältö keskitetään kapeammaksi, jotta sisältö on helpompi nähdä yhdellä vilkaisulla. Bannerin korkeutta ja leveyttä voidaan rajoittaa, jotta se ei vie liikaa tilaa suuremmilla näytöillä. Käyttäjän on hyvä nähdä muuta kuin pelkästään banneri, kun hän tulee sivulle.



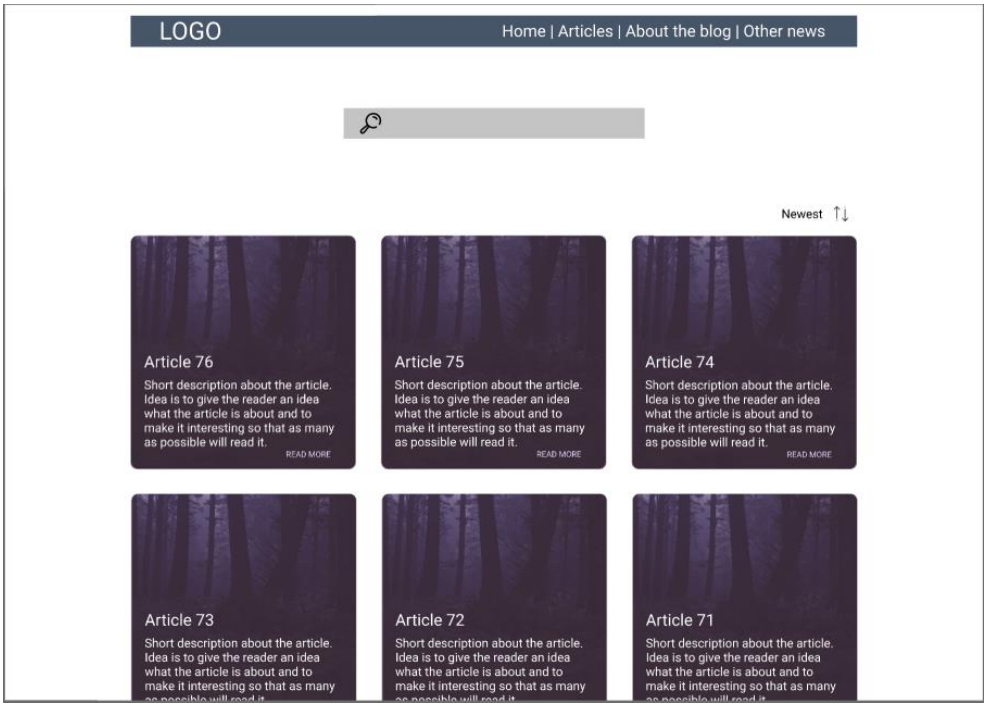
Kuvio 14: Etusivu väliaikaisella sisällöllä ja värimaailmalla

Kuviossa 15 kerrotaan blogista ja mikä sen tarkoitus on. Sivun banneriin voi myös laittaa kuvan, joka kuvaa blogia eli se voi olla kollaasi blogin tärkeimmistä hetkistä. Tälle sivulle tulee myös tieto, miten blogin hallitsijaan saa yhteyden. Ideana on lomake, jonka käyttäjä voi täyttää, mutta tätä toimintoa ei tämän kehittämistyön aikana toteuteta.



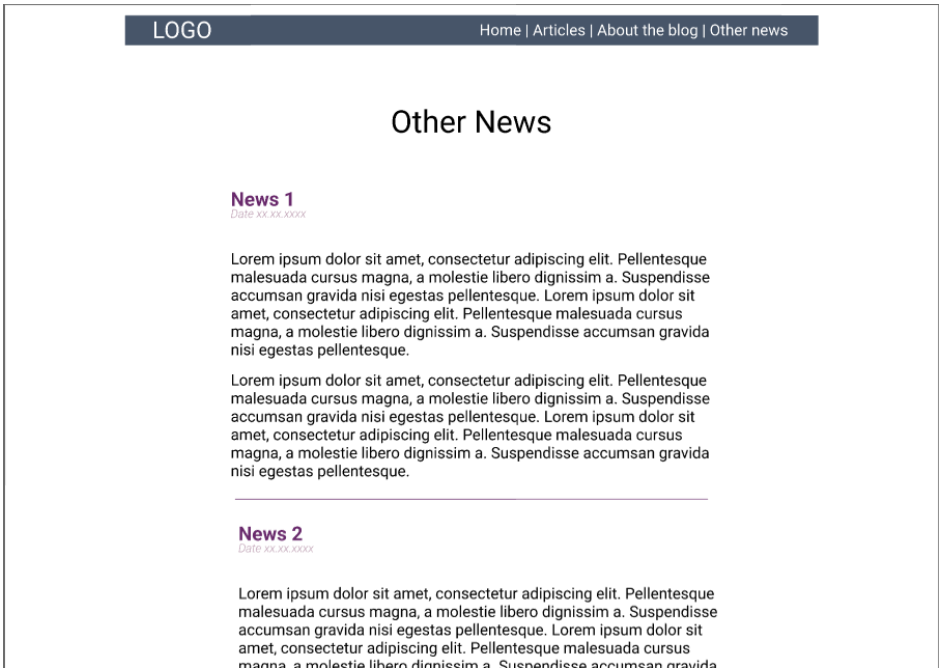
Kuvio 15: *About the blog* -sivu värimaailmalla ja kuvaavalla kuvalla

Kuviossa 16 on laitettu kaikki tarvittavat tyylit sivun elementeille. Blogikirjoituksien kortteihin on laitettu kuvan päälle liila läpinäkyvä kerros, jotta kaikilla blogipostauksilla näyttäisi olevan samanlainen teema ulkonäössä.



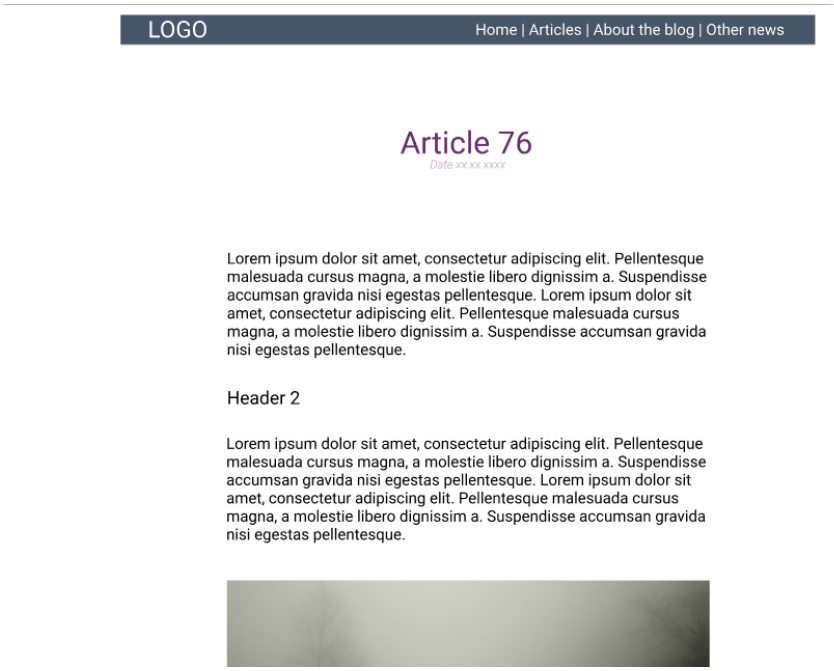
Kuvio 16: *Articles* -sivu väliaikaisella sisällöllä ja värimaailmalla

Kuviossa 17 on *other news* -sivun tyylitelty ulkonäkö ja väliaikaiseksi sisällöksi on laitettu ”Lorem ipsum” generaattorilla tehtyjä tekstin kappaleita. Sivun ulkonäön tarkentamisessa on myös päätetty, että uutisien väliin laitetaan ohut viiva, jotta uutiset erottuvat toisistaan tarkemmin. Uutisen otsikon alla on selkeästi, mutta himmeästi uutisen julkaisupäivä, jotta lukija tietää, miten ajankohtainen uutinen on. Olen päättänyt käyttää fonttia *Roboto,* joka on Googlen tarjoama ilmainen fontti. Kirjaimet ovat tavallista hieman korkeampia. Myös riviväliä suurennetaan, jotta ne erottuvat selkeästi toisistaan.



Kuvio 17: *Other news* -sivu väliaikaisella sisällöllä ja värimaailmalla

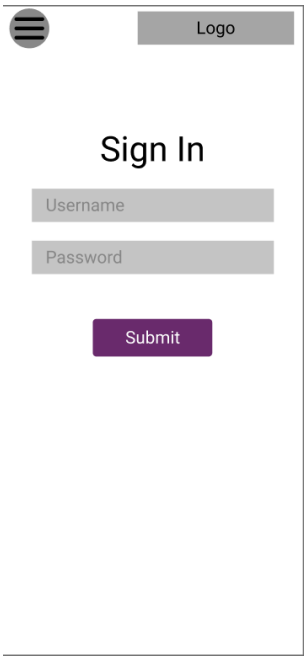
Kuviossa 18 on blogikirjoituksen sivun ulkonäkö. Se jätetään yksinkertaiseksi, jotta lukijan on helppo lukea sitä ja, jotta kirjoituksen kirjoittajalla on helppo järjestää eri kappaleet ja kuvat, miten haluaa.



Kuvio 18: Blogikirjoituksen ulkonäkö väliaikasella sisällölä ja värimaailmalla

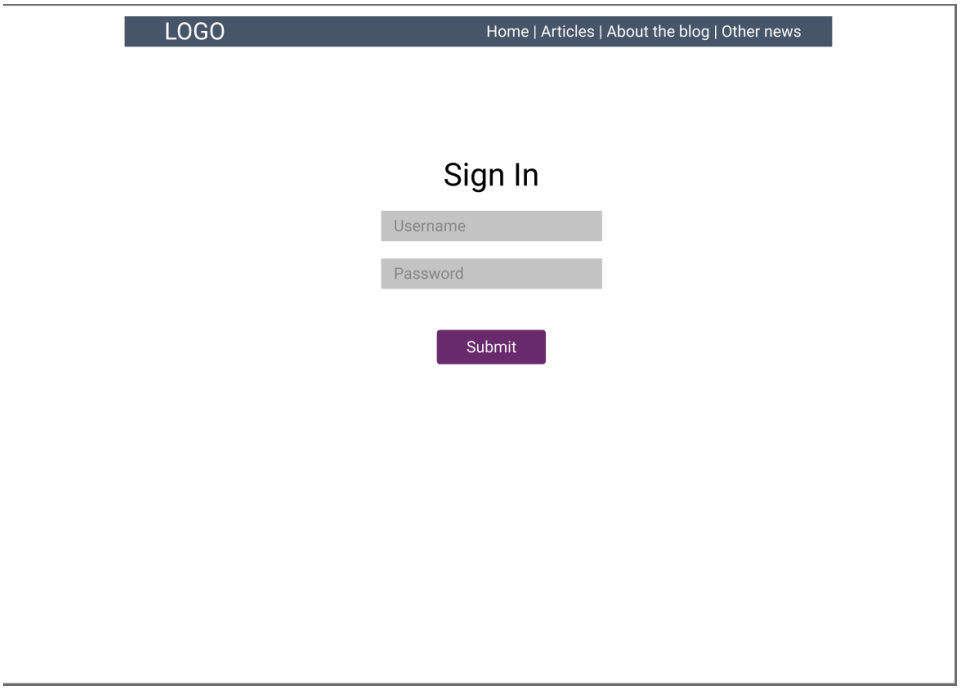
Prototypointiin käytettiin Figman prototypointityökalua. Sillä kokeiltiin, onko mobiilikäyttöliittymän suunnitelma intuitiivinen tavallisessa käytössä. Lyhyellä testaamisella päädyttiin siihen, että käyttöliittymä on yksinkertainen ja se muistuttaa käytöltään muita samantyyppisiä sivuja.

Hallintajärjestelmään on tarkoitus päästä vain sisään kirjautuneena. Tätä varten tarvitaan käyttöliittymään sivu, jonka kautta on helppo kirjautua sisään. Kuviossa 19 näkyy, miltä blogin *sign in* –sivu näyttää mobiililaitteella. Sivulla näkyy selkeästi kaksi tekstialuetta, *username* ja *password,* ja ohje, joka kertoo mitä tietoa tekstialueelle tulee laittaa. Tekstialueiden jälkeen on *submit* -nappi, jota painamalla käyttäjä pääsee kirjautumaan sisään.



Kuvio 19: *Sign in* -sivu mobiililaitteella

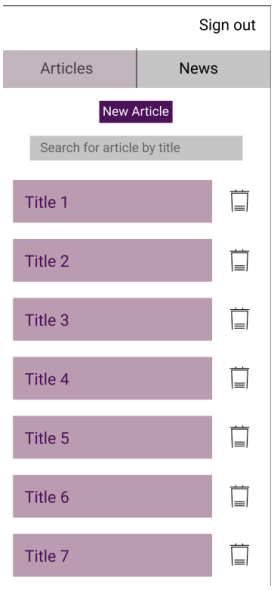
Kuviossa 20 on sama sivu suuremmalla näytöllä. Vaikka näyttö on suurempi, on tekstialueet keskitetty melkein saman kokeiseksi kuin mobiililaitteella. *Sign in* -sivua ei lisätä navigaatioon, sillä vain minulla tulee olemaan pääsy hallintajärjestelmään. Tämän kehittämistyön aikana vain minulle luodaan käyttäjätunnus, jonka takia käyttäjätunnuksien luontiin ei kehitetä omaa sivua.



Kuvio 20: *Sign in* -sivu suuremmalla näytöllä

Kun käyttäjä kirjautuu sisään, pääsee hän sivulle (kuvio 21 ja 22), jonka kautta hän voi helposti poistaa, lisätä ja muokata blogikirjoituksia. Sivulla on myös käytössä hakukenttä, jolla voi hakea blogikirjoituksia niiden otsikoiden perusteella. Blogikirjoituksen otsikkoa

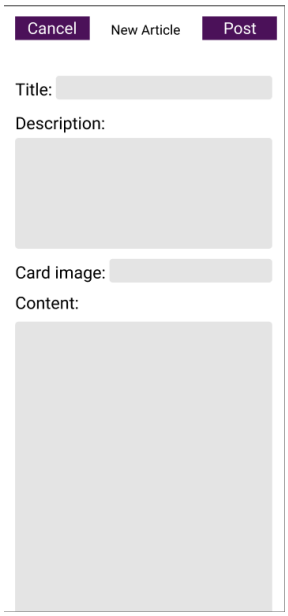
klikkaamalla avautuu kuvion 23 näköinen sivu, jossa blogikirjoituksien tietoja voidaan muokata. Otsikon vieressä on roskakorin näköinen ikoni, jota painamalla voi poistaa blogikirjoituksen. Sivun yläosasta voi vaihtaa artikkelien sijalle uutiset, joita voi myös poistaa ja luoda lisää.



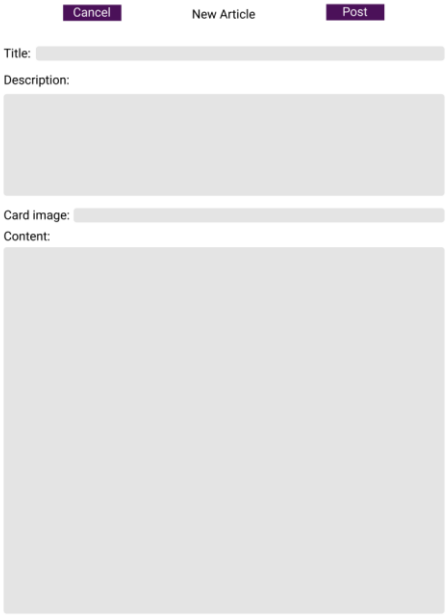
Kuvio 21: *Editarticles* -sivu mobiililaitteella

Kuvio 22: *Editarticles* - sivu isommalla näytöllä

Kun uuden blogikirjoituksen haluaa kirjoittaa, avautuu kuvion 23 näköinen sivu, jossa on jokaiselle blogikirjoituksen kohdalle oma tekstikenttä, otsikko, kuvaus, korttiin tuleva kuva ja blogikirjoituksen sisältö. Sivun yläreunassa on myös kaksi nappia, *cancel* ja *post*. Ensimmäisestä napista voi peruuttaa blogikirjoituksen kirjoittamisen ja toisesta blogikirjoitus julkaistaan. Kuviossa 24 on sama sivu, mutta isommalla näytöllä. Myös siinä sivun sisältö on keskitetty kapeammaksi kuin näytön koko, jotta käyttäjän on helpompi nähdä kokonaisuus, silmien ei tarvitse liikkua näytöllä laidasta laitaan.



Kuvio 23: *New article* -sivu mobiililaitteella



Kuvio 24: *New article* -sivu suuremmalla näytöllä

Uuden uutisen luominen onnistuu samalla tavalla kuin uuden blogikirjoituksen. Uusi sivu aukeaa, jossa on tekstialueet tarvittaville osille, tässä tapauksessa otsikko ja sisältö. Uutisista ei tarvitse kuvausta eikä kuvaa korttiin, sillä uutisista ei ole erillistä korttia. Kuten uuden blogikirjoituksen tekemisessä, myös uuden uutisen kirjoittaminen on keskitetty kapeammaksi kuin näytön leveys, jotta siitä on helppo nähdä kokonaisuus. Samalla näkee suurin pirtein, miten pitkää uutista on kirjoittamassa.

6 Kehittämistyön toteutus

Kehittämisympäristö on asennettu, joten sen tekemistä ei käydä läpi. Myös projekti on luotu GitHubiin, ja se on yhdistetty Visual Studio Codeen, jotta päivitetyn koodin voi helposti siirtää GitHubiin. Tässä kehittämistyössä käytetään Visual Studio Codea ohjelmointiin. Projekti on valmiiksi jaettu serveri- ja client-puoleen, ja sen tekemistä käydä läpi.

Kehittämistyössä hyödynnetään myös *Markdown* -merkintäkieltä. Markdown on tekstimuotoinen merkintäkieli, joka voidaan kääntää esimerkiksi HTML -muotoon. Selaimet lukevat HTML -muodossa olevaa tekstiä ja esittävät sen nettisivuna. (Markdown 2022) Kuviossa 25 näkyy, miltä teksti näyttää missäkin muodossa.

Markdown	HTML	Rendered Output
`# Heading level 1`	`<h1>Heading level 1</h1>`	Heading level 1
`## Heading level 2`	`<h2>Heading level 2</h2>`	Heading level 2

Kuvio 25: *Markdown* -merkintäkieli verrattuna HTML -kieleen (mukaillen Markdown guide 2022)

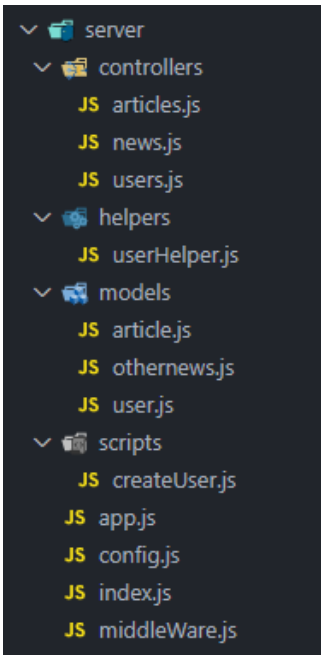
6.1 Palvelimen ja tietokannan kehittäminen

Palvelinpuolella eli sovelluksen backendissä hoidetaan tietokantaan yhteyden ottaminen. Palvelin hoitaa kommunikaation käyttöliittymän ja tietokannan välillä. Siellä myös käsitellään erilaisten lomakkeiden käsittelyt, ja käyttäjien kirjautuminen ja esimerkiksi käyttäjätunnuksien luominen.

Isommissa projekteissa palvelin kannattaa jakaa pienempiin osiin, etenkin, jos tietokannassa ja käyttöliittymässä halutaan käyttää monenlaista dataa. Kuviossa 26 on palvelimessa käytettävät tiedostot ja kansiot. Palvelin -tiedosto on nimetty serveriksi. Serveriin on luotu kansiot *models*, *controller* ja *scripts*. *Models* -kansiossa luodaan jokaiselle tietokannassa olevalle tietokokonaisuudelle objekti, joka sisältää tiedon, millä avaimella tieto on tietokannassa ja mitä tyyppiä tieto on.

Kuviossa 27 on tehty Mongoose -malli. Mongoose on kirjasto, jota käytetään datan mallintamiseen, mikä tekee palvelimen ja tietokannan välisestä käytöstä suoraviivaisempaa. Ensin määritellään kaavio, jonka rakenne on samantyyppinen kuin, miltä data näkyy tietokannassa, eli jokaisella tieto-osalla on oma nimi, kuten *title*, johon tallennetaan blogikirjoituksen otsikko. Blogikirjoitukselle on määritelty, että *title*, *description*, *image* ja *content* ovat pakollisia. *Release_date* ja *authorId* luodaan automaattisesti siinä vaiheessa, kun uusi blogikirjoitus luodaan. Kuviossa 27 on *article.js* tiedosto, joka on Models-kansiossa.

Kuviossa 26 on myös kansio *controllers*, jossa jaetaan API:n reitityksen pienempiin osiin ja omiin tiedostoihin. Tiedosto jaetaan samalla tavalla aiheittain, kuten myös Models-kansiossa olevat tiedostot. Pienempiin osiin jakaminen selkeyttää kehittäjälle, mitä tapahtuu missäkin tiedostossa ja koodi näyttää myös selkeämmältä.



Kuvio 26: Palvelimen tiedostojen hierarkia

```
// schema for article
const mongoose = require('mongoose')
const Schema = mongoose.Schema

const ArticleSchema = new Schema({
    title: {
        type: String,
        require: [true, 'Title required.']
    },
    description: {
        type: String,
        required: [true, 'Please add a short description about the article.']
    },
    image: {
        type: String,
        required: [true, 'Please add image.']
    },
    content: {
        type: String,
        require: [true, 'Content required.']
    },
    release_date: {
        type: Date,
        require: [true]
    },
    authorId: {
        type: Schema.Types.ObjectId
    },
    clickrate: {
        type: Number
    }
})

const Article = mongoose.model('articles', ArticleSchema)

ArticleSchema.set('toJSON', {
    transform: (document, returnedObject) => {
        returnedObject.id = returnedObject._id.toString()
        delete returnedObject._id
    }
})

module.exports = Article
```

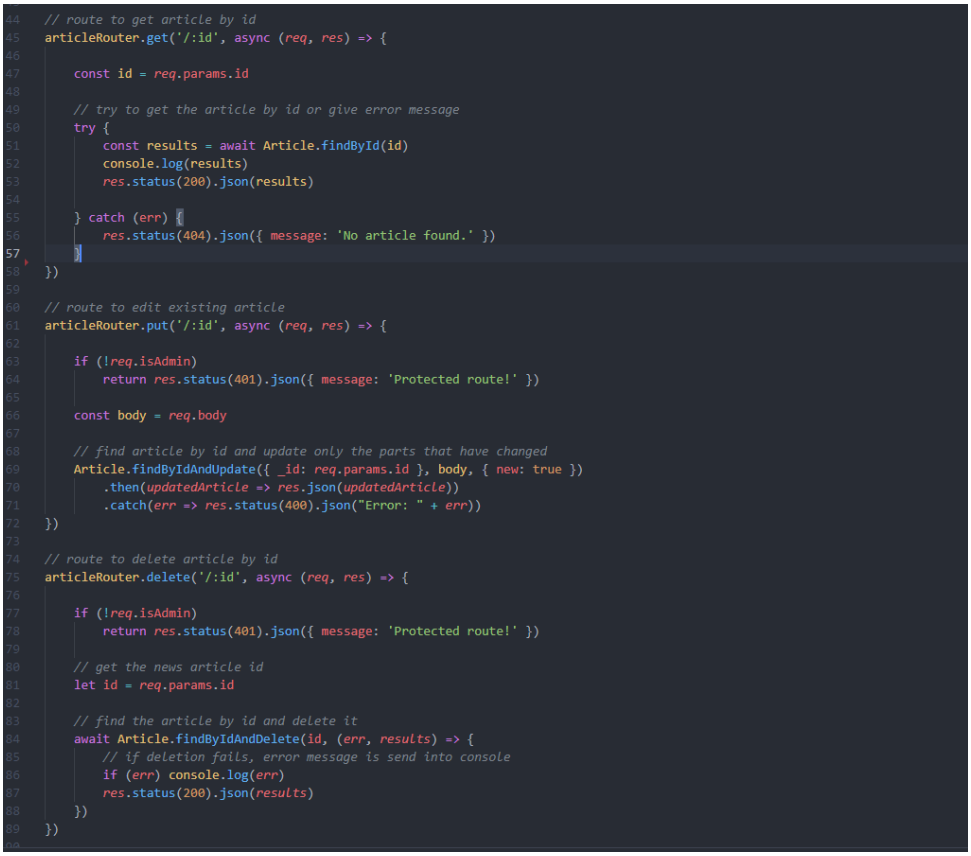
Kuvio 27: Blogikirjoituksen malli, jolla se tallennetaan tietokantaan

Kuvioissa 28 ja 29 on blogikirjoituksen *controller* -tiedosto. Reitityksien tarkoitus on kommunikoida käyttöliittymän ja tietokannan välillä. Kuviossa 28 reitti *articleRouter.get('/all')* hakee tietokannasta kaiken blogikirjoituksiin liittyvän datan ja lähettää sen lopuksi eteenpäin JSON -muodossa käyttöliittymälle käytettäväksi. Käyttöliittymän puolella tämä JSON -muodossa oleva data tyylitellään, jotta sitä on helpompi

lukea. Kaikkien reitityksien tarkoituksena on saada käyttöliittymän puolelta blogikirjoituksen *id*, ja tehdä tämän perusteella muokkauksia ja poistoja. Kaikissa metodeissa GET -metodia lukuun ottamatta katsotaan ensin, onko käyttäjä kirjautunut sisään. Näin estetään muiden pääsy lisäämään, muokkaamaan ja poistamaan tietokannassa olevaa dataa. Kuviosta 28 ja 29 näkyy, miten tämä on toteutettu. Kehittämistyössä käytetään *if* -lausetta, jolla katsotaan, onko käyttäjällä tokenia vai ei. Jos tokenia ei ole, käyttöliittymään palautuu virheviesti. Token tarkastetaan erillisessä tiedostossa kuvion 30 mukaisesti. Oikea tokeni on tallennettu *.env* tiedostoon, johon selaimessa olevaa tokenia verrataan. Sen täytyy täsmätä *.env* tiedossa olevaa, jotta reitit toimivat.

```
server > controllers > JS articles.js > ...
1   const articleRouter = require('express').Router()
2
3   // import mongoose model
4   const Article = require('../models/article')
5
6   // route to get all articles
7   articleRouter.get('/all', async (req, res) => {
8
9       // save all articles into a varibale that is send as a response to the request made in the frontend
10      const results = await Article.find()
11
12      res.status(200).json(results)
13  })
14
15  // route to make a new article post and save it into the database
16  articleRouter.post('/submit', async (req, res) => {
17
18      if (!req.isAdmin)
19          return res.status(401).json({ message: 'Protected route!' })
20
21      const body = req.body
22
23      // saving the data given in the frontend into variables given by the controller
24      const article = new Article({
25          title: body.title,
26          description: body.description,
27          content: body.content,
28          image: body.image,
29          release_date: new Date(),
30          authorId: req.token.id,
31          clickrate: 0
32      })
33
34      // try to save the information into the database or give error
35      try {
36          await article.save()
37          res.status(200).json(article)
38      } catch (err) {
39          console.error(err)
40          res.status(500).json(err)
41      }
42  })
```

Kuvio 28: Blogikirjoituksien GET- ja POST-reitit



```
// route to get article by id
articleRouter.get('/:id', async (req, res) => {

    const id = req.params.id

    // try to get the article by id or give error message
    try {
        const results = await Article.findById(id)
        console.log(results)
        res.status(200).json(results)

    } catch (err) {
        res.status(404).json({ message: 'No article found.' })
    }
})

// route to edit existing article
articleRouter.put('/:id', async (req, res) => {

    if (!req.isAdmin)
        return res.status(401).json({ message: 'Protected route!' })

    const body = req.body

    // find article by id and update only the parts that have changed
    Article.findByIdAndUpdate({ _id: req.params.id }, body, { new: true })
        .then(updatedArticle => res.json(updatedArticle))
        .catch(err => res.status(400).json("Error: " + err))
})

// route to delete article by id
articleRouter.delete('/:id', async (req, res) => {

    if (!req.isAdmin)
        return res.status(401).json({ message: 'Protected route!' })

    // get the news article id
    let id = req.params.id

    // find the article by id and delete it
    await Article.findByIdAndDelete(id, (err, results) => {
        // if deletion fails, error message is send into console
        if (err) console.log(err)
        res.status(200).json(results)
    })
})
```

Kuvio 29: Blogikirjoituksen GET-, PUT- ja DELETE-reititykset

```
const jwt = require('jsonwebtoken')

const tokenExtractor = (request, response, next) => {
    const authorization = request.get('authorization')
    if (authorization && authorization.toLowerCase().startsWith('bearer ')) {
        request.token = decodeToken(authorization.substring(7))
        if (request.token)
            request.isAdmin = true
    } else {
        request.token = null
    }
    next()
}

const decodeToken = (token) => {
    try {
        if (token)
            return jwt.verify(token, process.env.TOKEN_KEY)
        else
            return undefined
    } catch (err) {
        return undefined
    }
}

module.exports = {
    tokenExtractor
}
```

Kuvio 30: Tokenin tarkastus

Kuviossa 26 näkyville muille tiedostoille kansioissa *models* ja *controllers* tehdään samanlaiset mallit ja reititykset, jotta niihinkin saadaan käyttöliittymän puolelta yhteys. Reitityksille

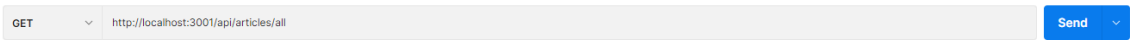
annetaan myös metodi, joka kertoo sille, mihin tarkoitukseen reitti on tehty. Tässä kehittämistyössä käytetään neljää eri metodia: GET, POST, PUT ja DELETE. GET -metodi hakee tietokannasta halutun datan, POST -metodi lisää tietokantaan dataa, PUT -metodi päivittää dataa ja DELETE -metodi poistaa dataa.

```
// Routes and mongoose
const express = require('express')
const app = express()
const config = require('./config')
const path = require('path')
const mongoose = require('mongoose')
const middleware = require('./middleWare')
const cors = require('cors') //im not sure this is needed

// parse json bodies sent by api clients
app.use(express.json())
app.use(express.urlencoded({
    extended: true
}));

app.use(middleware.tokenExtractor)

// routes to what controllers to use
const articleRouter = require('./controllers/articles')
app.use('/api/articles', articleRouter)

const newsRouter = require('./controllers/news')
app.use('/api/othernews', newsRouter)

const userRouter = require('./controllers/users')
app.use('/api/user', userRouter)

// connecting to database
mongoose.connect(config.MONGODB_URI, {});

var db = mongoose.connection;

// Making sure the connection is working
db.on("error", function () {
    console.log("Connection error!")
});

db.once("open", function () {
    console.log("Connection successful!");
});


app.use(cors())
```

Kuvio 31: *app.js* -tiedoston sisältö

Serverillä on *app.js* -niminen tiedosto (kuvio 31), jota voidaan ajatella serverin yhteiseksi tiedostoksi. Kuvioissa 28 ja 29 tehdyt reitityksen voidaan myös tehdä tähän tiedostoon. Kun palvelimen laittaa päälle, alkaa se lukemaan tästä tiedostosta koodin. Tiedostossa luodaan pääsy *controller* -tiedostoihin ja luodaan yhteys tietokantaan.

Metodien ja reitityksien toimimista testataan Postmanilla. Postmanilla voidaan testata API:n toimivuutta lähettämällä erilaisia kutsuja sen kautta, käyttämällä reitityksissä käytettyjä metodeja. Kutsu lähetetään GET -metodilla (kuvio 32) oikeeseen osoitteeseen ja vastauksena palautuu data JSON -muodossa (kuvio 33).



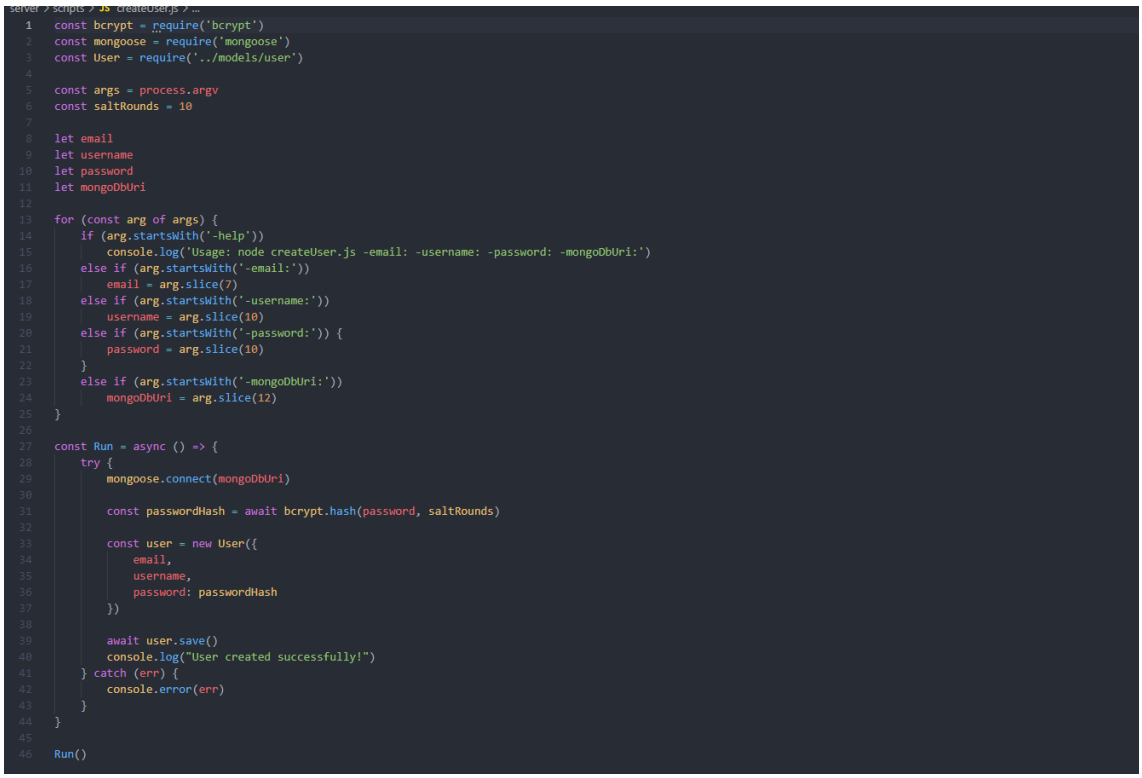
Kuvio 32: Postmanilla lähetetty kutsu GET -metodilla


```
[
    {
        "title": "The Attack on unrestricted Dashboard",
        "description": "There once was a website with a dashboard.\nWithout any authorization.\nNow there is a website with this article in it!",
        "image": "https://i.kym-cdn.com/entries/icons/mobile/000/021/007/ig9OoyenpxqdCQyABmOQBZDI0duHk2QZZmWg2Hxd4xo.jpg",
        "content": "# Why does your website need Autherization handeling?\n\nNot every website need authorization. But if your website has user generated content (admin or otherwise), it should also have some sort of authorization.\n\nExposing admin dashboards to the world wide web is a sure way to get your website filled with third party content that you didn't want on your website.\n\nAdditionally, exposing your admin dashboard to unauthorized personnel puts your website in risk of cross site scripting (XSS) attacks.\n\n\n\n\n## How do I protect my website from the internet?\n\nThere are many ways to integrate authorization to your website, but the best and surest way is to use third party authorization specialist made software like [0auth](https://auth0.com/)\n\nThis part of the blog is still under construction. Thank you for your understanding <3",
        "release_date": "2022-03-29T16:26:30.112Z",
        "authorId": "5f87ac56842e500d044853dd",
        "clickrate": 0,
        "__v": 0,
        "id": "62433336998009edd268967e"
    },
```


Kuvio 33: Kutsuun saatu vastaus Postmanilla

Tietokantaan luodaan käyttäjä käyttäen skriptiä (kuvio 34). Skripti ajetaan komentoikkunassa, jossa sille annetaan halutut tiedot. Tässä tapauksessa annetaan

sähköpostiosoite, käyttäjäni, salasana ja tietokannan osoite, joka on myös *.env* tiedostossa. Skripti luo käyttäjän tietokantaan ja samalla salaa salasanan.



```
server > scripts > JS createUser.js > ...
const bcrypt = require('bcrypt')
const mongoose = require('mongoose')
const User = require('../models/user')

const args = process.argv
const saltRounds = 10

let email
let username
let password
let mongoDbUri

for (const arg of args) {
    if (arg.startsWith('-help'))
        console.log('Usage: node createUser.js -email: -username: -password: -mongoDbUri:')
    else if (arg.startsWith('-email:'))
        email = arg.slice(7)
    else if (arg.startsWith('-username:'))
        username = arg.slice(10)
    else if (arg.startsWith('-password:')) {
        password = arg.slice(10)
    }
    else if (arg.startsWith('-mongoDbUri:'))
        mongoDbUri = arg.slice(12)
}

const Run = async () => {
    try {
        mongoose.connect(mongoDbUri)

        const passwordHash = await bcrypt.hash(password, saltRounds)

        const user = new User({
            email,
            username,
            password: passwordHash
        })

        await user.save()
        console.log("User created successfully!")
    } catch (err) {
        console.error(err)
    }
}

Run()
```

Kuvio 34: Skripti, mikä luo käyttäjän tietokantaan

Kuviossa 26 on tiedostot *.gitignore* ja *.env*, jotka auttavat estämään arkaluonteisten tietojen päätymistä internettiin kaikille nähtäväksi. Jotta palvelin pystyy yhdistämään tietokantaan, tarvitsee tämä tietynlaisen osoitteen, josta näkyy sekä käyttäjätunnus että salasana. Tällainen tieto on arkaluonteinen, sillä ei kenellä tahansa saa olla pääsyä suoraan tietokantaan. *.env* tiedostoon laitetaan tietokannan osoite ja siellä se tallennetaan erilliseen muuttujaan, jota käytetään muualla palvelimella. *.gitignore* on tiedosto, johon määritellään, mitä tiedostoja tai tiedostotyyppejä ei voi puskea Githubiin.

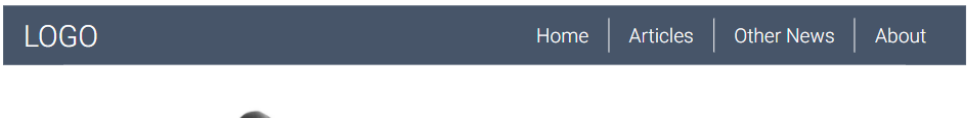
6.2 Käyttöliittymän kehittäminen

Käyttöliittymän kehittäminen on aloitettu tekemällä toimiva navigaatio, sekä mobiililaitteelle että isommalle näytölle. Ensin on luotu tarvittavat sivut, joiden välillä navigaation kautta pääsee, mutta sivuille ei luotu muuta sisältöä kuin sivun otsikko tai banneri, jotta voidaan nähdä toimiiko sivunvaihto.

Kuviossa 35 on käyttöliittymän käyttämät tiedostot. Päätiedostot on jaettu sivuihin ja komponentteihin. Komponentit ovat sellaisia osia, joita käytetään useammalla sivulla. Tämä ehkäisee saman koodin kirjoittamista useampaan kertaan. Esimerkiksi jokaisella sivulla on samanlainen alaviite, joten sen kirjoittaminen jokaiselle sivulle ei kannata. Kuviossa 35 on sivujen ja komponenttien tyyli -tiedostot.

Kuvio 35: Käyttöliittymän käyttämät kansiot ja tiedostot

Kuviossa 36 on navigaatio, joka on käytössä isommilla näytöillä. Navigaatio on luotu omaksi komponentiksi, eli navigaatiota ei tarvitse uudelleen luoda jokaiselle sivulle kuten perinteisessä HTML -sivussa. Tällaisten komponenttien käyttö nopeuttaa sivun latausta, sillä sivulla on vähemmän lataamista, kun ei tarvitse renderoida samaa komponenttia tai sivun osaa uudelleen joka kerta, kun osa sivun sisällöstä muuttuu.

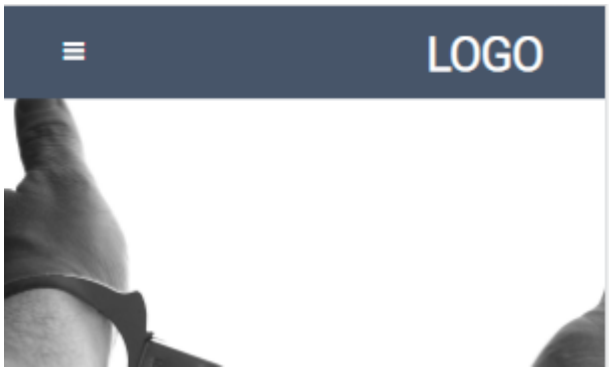


Kuvio 36: Navigaatio isommalla näytöllä

Kuviossa 37 on koodi, jolla navigaatio on kehitetty. Navigaation komponentin nimeksi on annettu *Navbar* ja se sisältää erilaiset muuttujat ja funktiot, jolla navigaatioon on lisätty toimintoja. Komponentti myös palauttaa sivulla käytetyt elementit. Elementit jakavat sivun eri osiin ja samalla kertovat selaimelle, mitä eri osia sen pitää renderoida. Elementeillä on erilaisia tageja, jotka kertovat selaimelle, minkälaista tietoa elementissä on. Elementillä <*i*> on tuotu ikoni sivulle, jolle on annettu toiminto, joka muuttaa ikonin ulkonäköä riippuen siitä, onko mobiililaitteella navigaatio auki tai kiinni. Kuviossa 38 navigaatiossa on kolme raitaa, jotka viittaavat siihen, että sitä painalla saa avattua navigaation. Tällaisen merkin näkee todella usein erilaisilla sivuilla. Kun tätä nappia painaa kuvion 37 koodissa *setToggleMenu*:n tieto muuttuu ja samalla se muuttaa minkälainen ikoni näytetään sivulla. Kun navigaatio on auki, näkyy kuvion 39 tapainen rasti, jota painalla navigaation saa takaisin kiinni.

```jsx
import '../styles/Navbar.css'
import { useState } from 'react'
import { Link } from 'react-router-dom'

// font-awesome css file needed so the fonts work in classname
import 'font-awesome/css/font-awesome.min.css'

import { MenuList } from './MenuList'

const Navbar = () => {

    const [toggleMenu, setToggleMenu] = useState(false)

    const menuList = MenuList.map(({ url, title }, index) => {
        return (
            <li key={index}>
                <Link to={url}>{title}</Link>
            </li >
        )
    })

    const handleClick = () => {
        setToggleMenu(!toggleMenu)
    }
    return (
        <nav>
            <div className='menu-icon' onClick={handleClick}>
                <i className={toggleMenu ? 'fa fa-times' : 'fa fa-bars'}></i>
            </div>
            <div className='logo'>
                <Link to="/homepage">
                    LOGO
                </Link>
            </div>
            <ul className={toggleMenu ? 'menu-links' : 'menu-links close'} onClick={() => setToggleMenu(prevState => !prevState)}>
                {menuList}
            </ul>
        </nav >
    )

}

export default Navbar
```

Kuvio 37: Navigaation toteutus



Kuvio 38: Navigaatio mobiililaitteella kiinni

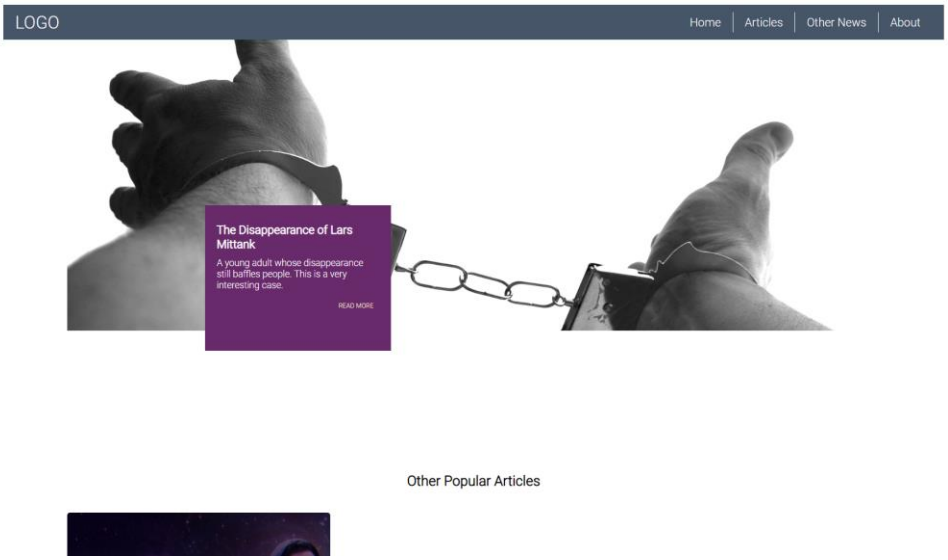
Kuvio 39: Navigaation mobiililaitteella auki

Navigaation ulkonäkö muutetaan kuvion 38 kaltaiseksi muuttamalla sen tyyliä. Kuviossa 40 oleva *@media* kohdassa tyylille annetaan ohjeet, jossa se tarkistaa käytetyn näytön leveyden ja muuttaa tiettyjä sivulla olevia tyylejä. Tässä tapauksessa navigaation ulkonäkö muuttuu mobiilinäkymäksi, kun näytön maksimi leveys on 600 pikseliä.

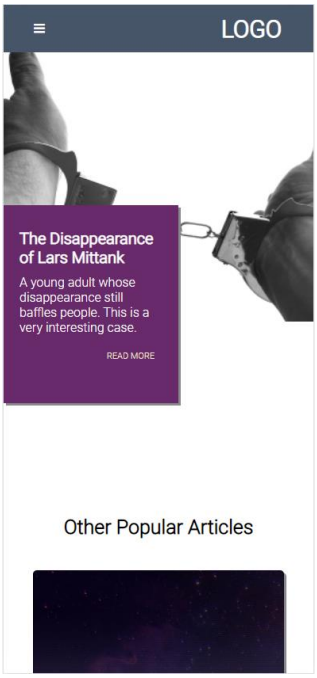
```css
@media screen and (max-width: 600px) {
    nav {
        background-color: transparent;
        color: black;
        height: 60px;
    }
    .menu-icon {
        display: block;
        background-color: #475569;
        padding: 15px;
        border-radius: 50%;
        position: relative;

    }
    .menu-links {
        flex-direction: column;
        justify-content: left;
        align-items:flex-start;
        width: 100%;
        position: absolute;
        background-color: #475569;
        padding: 20px;
        right: 0;
        top: 60px;
        border-bottom-right-radius: 150px;
        transition: all 0.5s ease-in-out;
    }

    .menu-links.close {
        right: 100%;
        transition: all 0.5s ease-in-out;
    }

    .menu-links li {
        padding: 20px;
        margin-bottom: 20px;
        width: 100%;
    }
    .menu-links li a {
        border-right: none;
```

Kuvio 40: Navigaation tyyli -tiedosto

Etusivulle on haettu blogikirjoituksien tietoja kuvion 43 mukaisesti, eli käyttäen *axios* -komentoa ja käyttämällä palvelimen puollella määritettyä osoitetta *axios.get('/api/articles/all')*. Komennolla on haettu tietokannasta kaikki blogikirjoitukset, jonka jälkeen ne täytyy järjestää uudelleen, jotta banneriin saadaan uusin blogikirjoitus. Järjestelyä varten on käytetty *sort()* – funktiota. Järjestelyn jälkeen ensimmäinen blogikirjoitus on aina uusin. Kuvioista 41 ja 42 näkee, kuinka uusin blogikirjoitus esitetään eri tavalla kuin muut blogikirjoitukset. Tämä suuntaa sivulla kävijän huomion uusimpaan blogikirjoitukseen.



Kuvio 41: Etusivu leveällä näytöllä, joka on ottanut tietokannasta tiedot blogikirjoituksia



Kuvio 42: Etusivu mobiililaitteella

```
import axios from 'axios'
import { useEffect, useState } from 'react'
import { Link } from 'react-router-dom'
import '../styles/Homepage.css'

import ArticleCard from '../components/ArticleCard'

const Homepage = () => {

    const [articles, setArticles] = useState([])

    // get all articles from the database and change the order from newest to oldest
    const getArticles = () => {
        axios.get('/api/articles/all')
            .then((response) => {
                console.log(response.data)
                const allArticles = response.data
                allArticles.sort((a, b) => {
                    return Date.parse(b.release_date) - Date.parse(a.release_date)
                })
                setArticles(allArticles)
            })
    }
    useEffect(() => getArticles(), [])


    return (
        <div className='homepage'>
            <div className='banner'>
                <img className='bannerimage' src={handcuffed} alt='newest article pic' />
                <div className='latestarticle'>
                    {articles.length >= 1 && <div className='latestarticlecontent'>
                        <h3>{articles[0].title}</h3>
                        <p>{articles[0].description}</p>
                        <Link className="readmorelink" to={`article/${articles[0].id}`} >READ MORE</Link>
                    </div>}
                </div>
            </div>
            <div className='popularcontent'>
                <h2>Other Popular Articles</h2>
                <div className='populararticles'>
                    {articles.slice(1, 3).map(article =>
                        <ArticleCard key={article.id} article={article} />
                    )}
                </div>
            </div>
        </div >
    )
}

```

Kuvio 43: Etusivun koodi

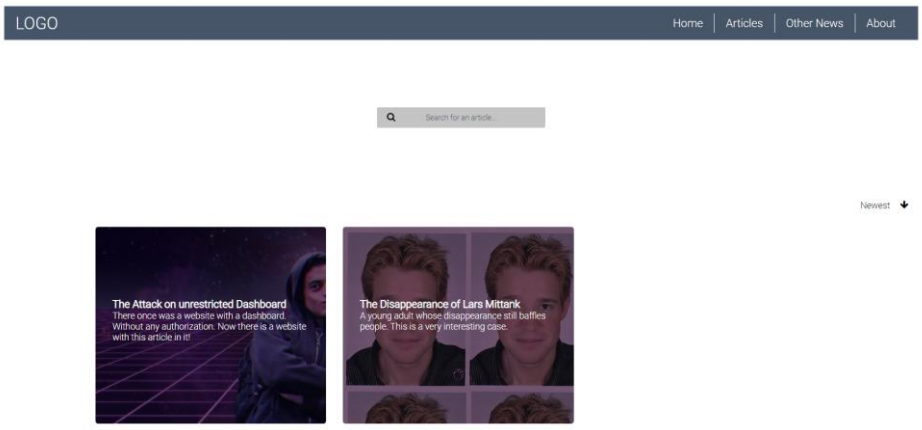
Jokaisessa blogikirjoituksessa on tietokannassa varattu avain, joka laskee, kuinka monesti blogikirjoitusta on klikattu, jotta etusivulle saadaan suosituimmat blogikirjoitukset uusimman

blogikirjoituksen lisäksi. Blogikirjoitukset järjestetään tämän avaimen perusteella suositummasta epäsuositumpiin ja siitä esitetään kuusi ensimmäistä blogikirjoitusta etusivulla.

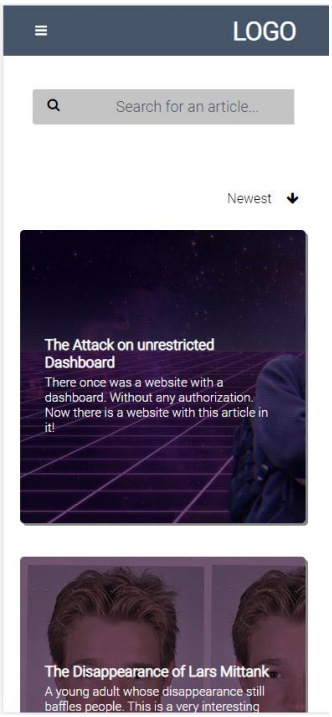
Suositummat blogikirjoitukset esitetään kortteina, jossa on käytetty komponenttia hyväksi (kuvio 44). Jokaiseen korttiin tulee blogikirjoituksesta samat kentät tietokannasta, joten kortille pitää antaa vain tieto, minkä blogikirjoituksen tietoja esitetään missäkin kortissa. Tämä on tehty antamalla, jokaiselle kortille blogikirjoituksen *id*, joka on jokaisella kirjoituksella ainutlaatuinen. Komponentti esittää blogikirjoituksen otsikon ja lyhyen kuvauksen blogikirjoituksen sisällöstä (kuvio 45 ja kuvio 46).

```jsx
import { Link } from 'react-router-dom'

const ArticleCard = ({ article }) => {
    return (
        <Link to={`article/${article.id}`
        }>
            <div
                style={{
                    backgroundImage: `linear-gradient(#413547, #413547), url(${article.image})`
                }}
                className='articlebanner'
                key={article.id}
            >
                <div className='cardtext'>
                    <h3>{article.title}</h3>
                    <p>{article.description}</p>
                </div>
            </div>
        </Link>
    )
}

export default ArticleCard
```

Kuvio 44: Blogikirjoituksen kortin komponentti



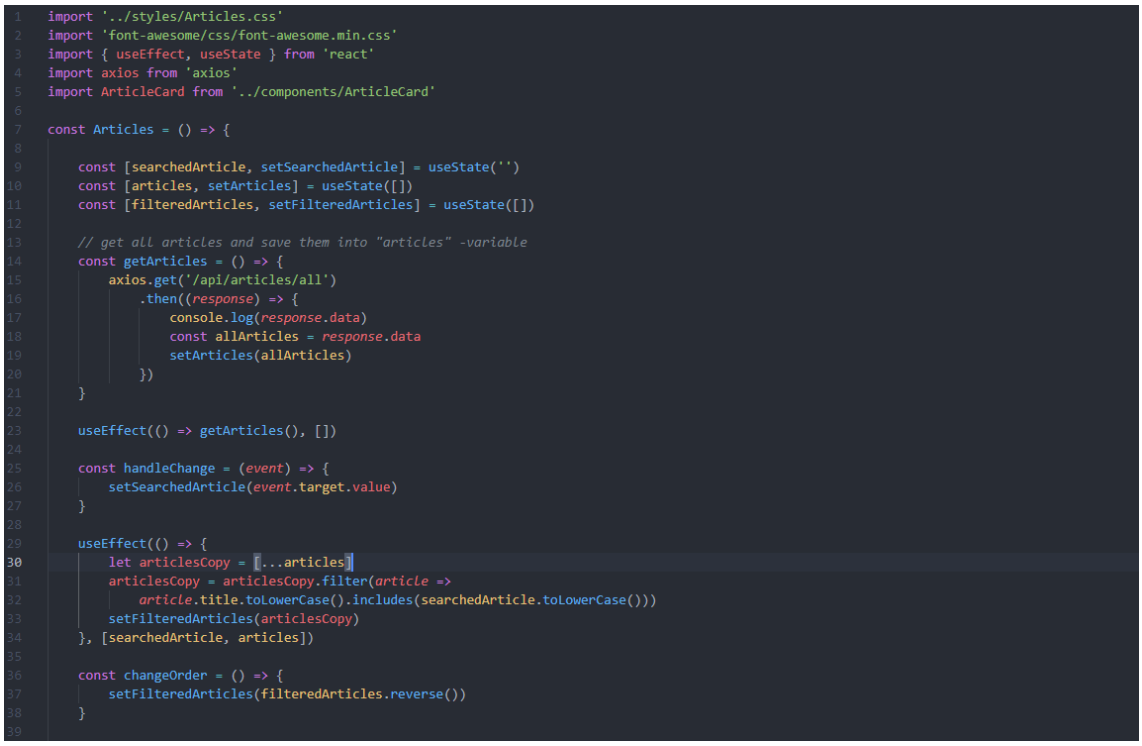
Kuvio 45: *Articles* - sivu, joka hakee tietokannasta kaikki blogikirjoitukset



Kuvio 46: *Articles* -sivu mobiililaitteella

Articles -sivulla haetaan samalla tavalla kaikki blogikirjoitukset tietokannasta kuin etusivullakin. Blogikirjoituksien lisäksi sivulla on myös hakutoiminto, jolla voi hakea haluamansa blogikirjoituksen sen otsikon perusteella. Jotta haku olisi mahdollisimman tehokas, on käytetty hakusanan perusteella olevaa suodattavaa hakua (kuvio 47). Hakutoiminto on kehitetty tekemällä alkuperäisestä taulukosta kopio, jossa kaikki blogikirjoitukset ovat. Muuttuja, johon kopio taulusta on tehty *articlesCopy*. Tämän jälkeen kopiota suodatetaan saadulla hakusanalla. Jäljelle jääneet blogikirjoitukset tallennetaan *filteredArticles* -nimiseen muuttujaan, jota käyttöliittymä käyttää esittääkseen hakusanaa vastaavat blogikirjoitukset. Tämä tarkoittaa sitä, että näytettävät blogikirjoitukset päivittyvät sitä mukaan, kun hakusana päivittyy. Esimerkiksi hakusanana voi olla pelkkä tavu ja kaikki blogikirjoitukset, jossa tämä tavu on otsikossa, suodatetaan tietokannasta sivulle. Tällaisella haulla käyttäjän ei tarvitse tietää tarkkaan hakemansa blogikirjoituksen otsikkoa.

Blogikirjoitukset esitetään taulukossa (kuvio 47). Se kuinka monta blogikirjoituksen korttia on rivissä, riippuu näytön leveydestä. Kun näytön leveys on niin kapea, että vain yksi kortti mahtuu riville, muutetaan sivun tyylittely sopivaksi mobiililaitteelle. Yhdelle riville mahtuu kuitenkin enintään 3 korttia, jotta sivun sisältö pysyy keskittyneenä.



```
import '../styles/Articles.css'
import 'font-awesome/css/font-awesome.min.css'
import { useEffect, useState } from 'react'
import axios from 'axios'
import ArticleCard from '../components/ArticleCard'

const Articles = () => {

    const [searchedArticle, setSearchedArticle] = useState('')
    const [articles, setArticles] = useState([])
    const [filteredArticles, setFilteredArticles] = useState([])

    // get all articles and save them into "articles" -variable
    const getArticles = () => {
        axios.get('/api/articles/all')
            .then((response) => {
                console.log(response.data)
                const allArticles = response.data
                setArticles(allArticles)
            })
    }

    useEffect(() => getArticles(), [])

    const handleChange = (event) => {
        setSearchedArticle(event.target.value)
    }

    useEffect(() => {
        let articlesCopy = [...articles]
        articlesCopy = articlesCopy.filter(article =>
            article.title.toLowerCase().includes(searchedArticle.toLowerCase()))
        setFilteredArticles(articlesCopy)
    }, [searchedArticle, articles])

    const changeOrder = () => {
        setFilteredArticles(filteredArticles.reverse())
    }
```

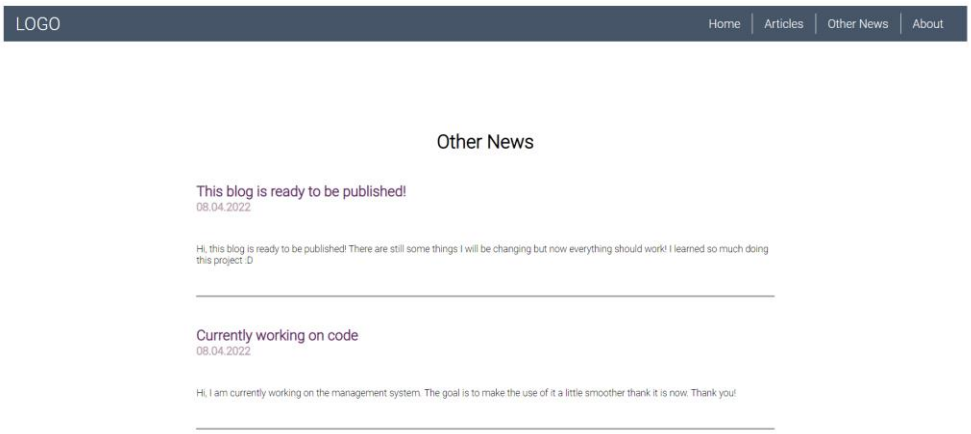
Kuvio 47: *Articles* -sivun koodi hakutoiminnolla

Blogikirjoitukset laitetaan taulukkomuotoon, muokkaamalla sivun tyyliä. Kuviossa 48 on *articles* -sivun tyylittely. Taulukon saa tehtyä laittamalla blogikirjoituksen *display* ominaisuudeksi taulukon ja taulukon koko on määritelty *grid-template-columns*:in avulla. Sillä on määritelty, kuinka monta yhdellä rivillä on blogikirjoituksen korttia ja *gap*:in avulla voidaan määritellä, miten iso väli korttien välillä on. Samalla määritellään, miten iso yksi kortti on. Kuviossa 48 on määritelty 3 korttia yhdellä rivillä kuvion 45 mukaisesti.

```
.articles {
    margin-top: 100px ;
    width: 100%;
    justify-content: center;
}
.searchbar {
    padding: 20px;
    justify-content: center;
    align-items: center;
    width: 100%;
}
.searchbar i {
    position: absolute;
}

i {
    padding: 10px;
    color: blue;
    min-width: 50px;
    text-align: center;
}

.searchbar input {
    width: 300px;
    color: black;
    border: none;
    background: #C4C4C4;
    text-align: center;
    padding: 10px;
    border-radius: 3px;
}
}

::placeholder {
    color: #475569;
}

.articlecontent {
    margin-top: 50px;
    width: 70%;
    height: auto;
    margin: auto;
    padding-top: 100px;
}

.allarticles {
    text-align: left;
    width: 100%;
    padding: 0px;
    display: grid;
    grid-template-columns: repeat(3, 1fr);
    gap: 20px;
    grid-auto-rows: minmax(100px, auto);
}
```

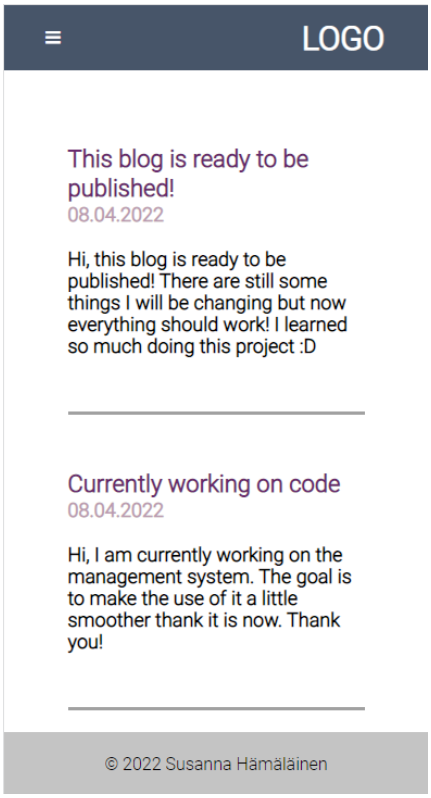
Kuvio 48: Osa *articles* -sivun tyylitiedostosta

Other news -sivulle haetaan uutiset samalla tavalla kuin blogikirjoitukset, vain osoite muuttuu. Uutisten esittelyyn käytetään myös komponenttia, jossa määritellään, että tietokannasta näytetään tiedot otsikko, päivämäärä ja uutinen. Kuviossa 49 sivun sisältö on kapeampi kuin näyttö ja tekstin kokoa ja rivivälin kokoa on suurennettu, jotta sitä olisi helpompi lukea.



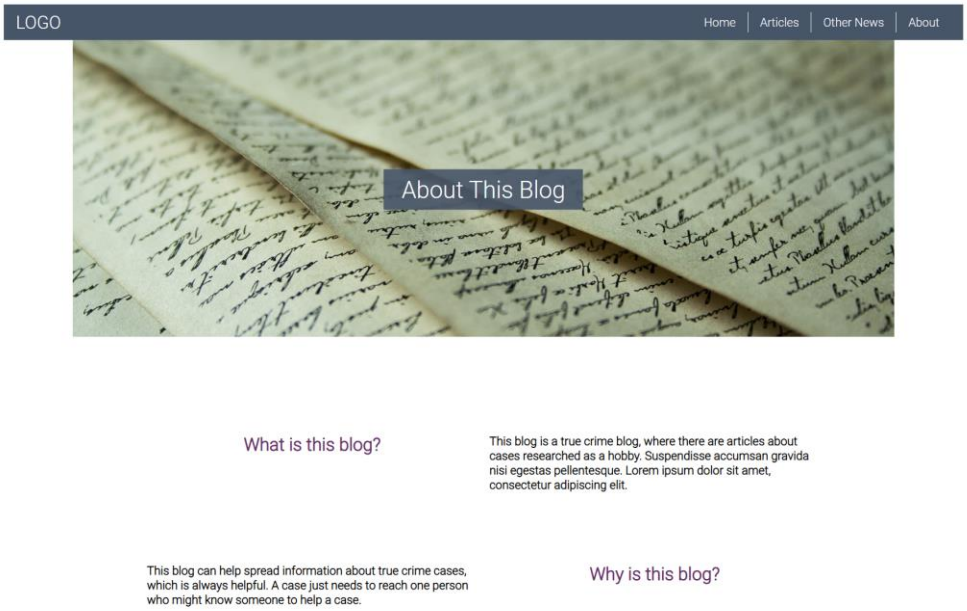
Kuvio 49: *Other news* -sivu

Kuviossa 50 *other news* -sivu mobiililaitteella näyttää paljon tiheämmältä. Olen toteutuksessa olettanut, että mobiililaitteella lukija on lähempänä näyttöä, joten sen lukeminen on myös helpompaa, jos sisältö on tiiviimmin kuin isommalla näytöllä.



Kuvio 50: *Other news* -sivu mobiililaitteella

About blog -sivu on kovakoodattu, eli se ei hae sisältö tietokannasta (kuvio 51). Kovakoodattu teksti tarkoittaa, että teksti löytyy koodista suoraan ja se ei päivity muuten kuin menemällä päivittämään koodia (kuvio 52). Tämän sivun sisältö ei tule aktiivisesti muuttumaan, joten päädyin tähän ratkaisuun.



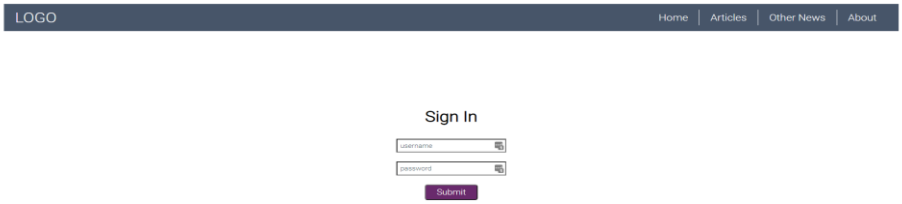
Kuvio 51: *About the blog* -sivu

```
import '../styles/About.css'
import banner from '../images/aboutpagebanner.jpg'

const About = () => {
    return (
        <div className='aboutpagewrapper'>
            <div className='aboutbanner'>
                <img src={banner} alt="handcuffed"></img>
                <p>About This Blog</p>
            </div>
            <div className='about-content'>
                <div className='whatblog'>
                    <h3>What is this blog?</h3>
                    <p>This blog is a true crime blog, where there are articles about cases researched as a hobby.

                        Suspendisse accumsan gravida nisi egestas pellentesque. Lorem ipsum dolor sit amet, consectetur adipiscing elit. </p>
                </div>
                <div className='whyblog'>
                    <h3>Why is this blog?</h3>
                    <p>This blog can help spread information about true crime cases, which is always helpful. A case just needs to reach one person who might know someone to help a case.</p>
                </div>
            </div>
        </div>
    )
}

export default About
```

Kuvio 52: *About the blog* -sivun kovakoodattu sisältö

6.3 Hallintajärjestelmän kehittäminen

Ensimmäisenä kehitin *sing in* -sivun (kuvio 53). Sivulla on kaksi teksti kenttää, joiden tiedot lähetetään palvelimen puolelle tarkistettavaksi. Kuviossa 54 reitti ottaa vastaan käyttöliittymästä lähetetyn datan ja etsii tietokannasta syötetyn käyttäjän. Jos tietokannasta löytyy annettu käyttäjä, verrataan seuraavaksi tietokannassa olevaa salattua salasanaa annettuun salasanaan. Jos käyttäjä on oikein, siirretään sisään kirjattu käyttäjä kuvion 56 mukaiselle *editarticles* -sivulle käyttäen Reactin tarjoamaa toimintoa *useHistory*, jolla voidaan siirtää käyttäjä tietylle sivulle antamalla sivun osoite. Jos käyttäjätunnus on väärin tai sitä ei ole olemassa lähetetään käyttöliittymään virheviesti liittyen epäonnistumiseen kirjautumiseen. Kirjautumisesta tallentuu token, joka on voimassa 28 päivää, eli niin kauan kuin se on voimassa, ei tarvitse erikseen kirjautua sisälle, vaan kirjautuminen on selaimen muistissa.



Kuvio 53: Hallintajärjestelmän *sign in* -sivu

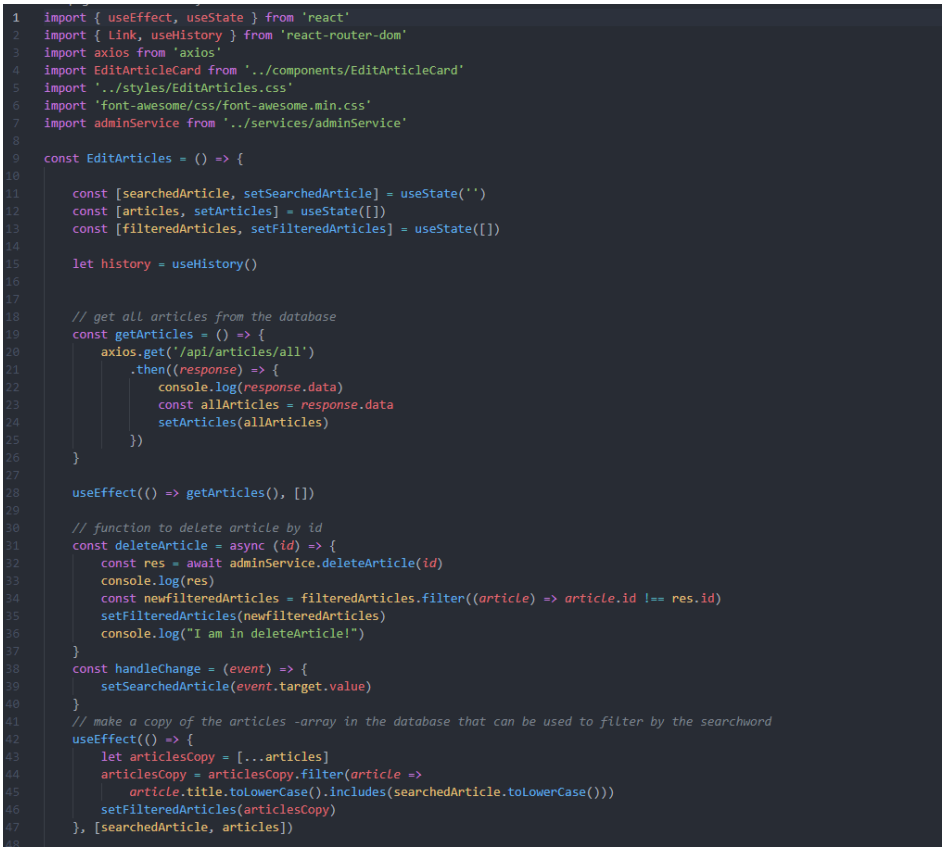
```
const bcrypt = require('bcrypt')
const userRouter = require('express').Router()
const jwt = require('jsonwebtoken')

// import mongoose model
const User = require('../models/user')


userRouter.post('/signin', async (req, res) => {
    const body = req.body

    let user
    // regex -> case insensitive
    if (body.username)
        user = await User.findOne({ username: { $regex: body.username, $options: 'i' } })
    else
        user = null

    // result is true or false
    const correctCredentials = user === null
        ? false
        : await bcrypt.compare(body.password, user.password)

    if (!correctCredentials) {
        res.status(401).json({ message: "Credentials incorrect!" })
        return
    }

    // create token
    const token = jwt.sign({
        username: user.username,
        id: user._id
    },
        process.env.TOKEN_KEY, {
        expiresIn: '28d'
    })

    res.status(200).send({ token })
})

module.exports = userRouter
```

Kuvio 54: Sisään kirjautuvan käyttäjän tarkistus

Editarticles -sivulle (kuvio 55) olen jättänyt navigaation, vaikka olin jättänyt sen suunnitteluvaiheessa pois. Kehittämisvaiheessa tuli ilmi, että navigaatiosta on hyötyä myös hallintajärjestelmän puolella. Hallintajärjestelmällä on kuitenkin vielä oma pieni navigaation, jonka avulla voi vaihtaa näkymää blogikirjoituksien ja uutisten välillä. Tämä pieni navigaatio on tehty samalla tavalla kuin blogin tavallinen navigaatio, eli linkataan suoraan uudelle sivulle, mutta vain uudet sivun osat ladataan uudelleen.

Kuvio 55: *Editarticles* -sivu

Blogikirjoituksen poistoa varten tarvitaan poistettavan blogikirjoituksen *id*. *Editarticles* -sivu käyttää myös omaa komponenttia, jonka avulla kaikki tietokannassa olevat blogikirjoitukset listataan, samalla tämä komponentti saa blogikirjoituksen *id*:n. Funktio, joka poistaa blogikirjoituksen tietokannasta on yhdistetty roskakori -ikoniin. Kun tätä painetaan, funktio kutsuu blogikirjoitukseen liittyvää DELETE – reittiä deleteArticle -funktion avulla (kuvio 56) ja poistaa tämän tietokannasta. Poiston jälkeen sivulle suodatetaan kaikki muut blogikirjoitukset poistettua blogikirjoitusta lukuun ottamatta.

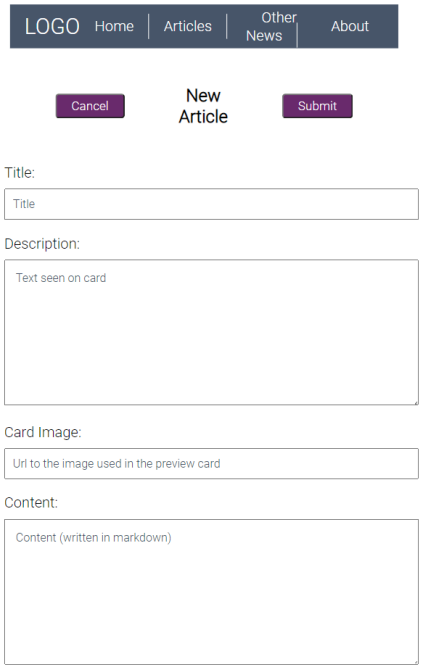


```
import { useEffect, useState } from 'react'
import { Link, useHistory } from 'react-router-dom'
import axios from 'axios'
import EditArticleCard from '../components/EditArticleCard'
import '../styles/EditArticles.css'
import 'font-awesome/css/font-awesome.min.css'
import adminService from '../services/adminService'

const EditArticles = () => {

    const [searchedArticle, setSearchedArticle] = useState('')
    const [articles, setArticles] = useState([])
    const [filteredArticles, setFilteredArticles] = useState([])

    let history = useHistory()


    // get all articles from the database
    const getArticles = () => {
        axios.get('/api/articles/all')
            .then((response) => {
                console.log(response.data)
                const allArticles = response.data
                setArticles(allArticles)
            })
    }

    useEffect(() => getArticles(), [])

    // function to delete article by id
    const deleteArticle = async (id) => {
        const res = await adminService.deleteArticle(id)
        console.log(res)
        const newfilteredArticles = filteredArticles.filter((article) => article.id !== res.id)
        setFilteredArticles(newfilteredArticles)
        console.log("I am in deleteArticle!")
    }
    const handleChange = (event) => {
        setSearchedArticle(event.target.value)
    }
    // make a copy of the articles -array in the database that can be used to filter by the searchword
    useEffect(() => {
        let articlesCopy = [...articles]
        articlesCopy = articlesCopy.filter(article =>
            article.title.toLowerCase().includes(searchedArticle.toLowerCase()))
        setFilteredArticles(articlesCopy)
    }, [searchedArticle, articles])
```

Kuvio 56: Osa *editarticles* -sivun koodi

Sivulla on myös uloskirjautumista varten *sign out* -nappi, joka poistaa tokenin selaimen paikallisesti muistista. Koska tokenia ei ole, ei käyttäjä myöskään pääse hallintajärjestelmään, eli hänet uudelleenohjataan *sign in* -sivulle. Kun uusi blogikirjoitus halutaan tehdä, painetaan *new article* -nappia. Tämä linkittää taas uudelle sivulle (kuvio 57).

Uuteen blogikirjoitukseen tarvitaan otsikko, kuvaus, kortin kuva ja sisältö (kuvio 57). Sisältö kirjoitetaan Markdown -kielellä. Jokaisen tekstikentän sisältö tallennetaan omaa muuttujaan, jotka lähetetään tietokantaan POST -reitityksellä (kuvio 58). Kun uusi blogikirjoitus lähetetään tietokantaan, myös käyttäjä uudelleenohjataan uuden blogikirjoituksen sivulle käyttäen edellä mainittua useHistory -toimintoa.



Kuvio 57: *New article* -sivun lomake

```
const NewArticle = () => {

    const [content, setContent] = useState('')
    const [title, setTitle] = useState('')
    const [description, setDescription] = useState('')
    const [cardImage, setCardImage] = useState('')

    const history = useHistory()

    // handlers to handle every change in the input fields and save them to their variables
    const getTitleHandler = (e) => {
        const givenTitle = e.target.value
        console.log(givenTitle)
        setTitle(givenTitle)
    }

    const getDescHandler = (e) => {
        const givenDesc = e.target.value
        console.log(givenDesc)
        setDescription(givenDesc)
    }

    const getImageHandler = (e) => {
        const givenImageUrl = e.target.value
        console.log(givenImageUrl)
        setCardImage(givenImageUrl)
    }

    const getContentHandler = (e) => {
        const givenContent = e.target.value
        console.log(givenContent)
        setContent(givenContent)
    }

    // function that submits article to be posted into the database
    const submitArticle = async () => {
        // try catch
        console.log("I am in submitArticle")
        const res = await adminService.createArticle({
            title,
            content,
            description,
            image: cardImage
        })

        history.push(`/article/${res.id}`)
    }
```

Kuvio 58: Koodi, mikä tallentaa annetut arvot muuttujiin

Eri reitityksen on kasattu erikseen yhteiseen tiedostoon (kuvio 59). Tästä on hyötyä, jos jonkun reitityksen osoite muuttuu. Jos osoite muuttuu, minun ei tarvitse erikseen mennä

jokaiseen tiedostoon muuttamaan osoitetta, vain yhteen, josta eri sivut hakevat osoitteen erikseen.

```
import axios from 'axios'

const getConfig = () => {
    const token = localStorage.getItem('token')
    let config = {}
    if (token) {
        config = {
            headers: { Authorization: `bearer ${token}` }
        }
    }

    return config
}

const createArticle = async (data) => {
    const res = await axios.post('api/articles/submit', data, getConfig())
    return res.data
}

const patchArticle = async (id, data) => {
    const res = await axios.put(`/api/articles/${id}`, data, getConfig())
    return res.data
}

const deleteArticle = async (id) => {
    const res = await axios.delete(`/api/articles/${id}`, getConfig())
    return res.data
}

const createNews = async (data) => {
    const res = await axios.post('api/othernews/submit', data, getConfig())
    return res.data
}

const deleteNews = async (id) => {
    const res = await axios.delete(`/api/othernews/${id}`, getConfig())
    return res.data
}

// eslint-disable-next-line import/no-anonymous-default-export
export default {
    createArticle,
    patchArticle,
    deleteArticle,
    createNews,
    deleteNews
}
```

Kuvio 59: Tiedosto, johon reitityksen on kasattu käyttöliittymän puolella

6.4 Sovelluksen testaaminen ja julkaiseminen

Sovellus on lähetetty parille henkilölle testattavaksi, jotta voin olla varma, että heillä ei, esimerkiksi ole pääsyä hallintajärjestelmään. Vain yhdellä testaajista tuli esille ongelmia blogikirjoituksen sivun kanssa, mutta ongelma ei ollut suoraan blogin puolella, vaan testaajan käyttöjärjestelmän asetuksissa.

Testaajat testasivat verkossa toimivaa versiota. Olen julkaissut blogin Herokun kautta. Heroku on vain yksi monesta mahdollisesta alustasta, jonka kautta omia web -sovelluksia voi julkaista. Herokuun olen myös laittanut *.env* tiedossa olevat arvot, sillä ilman tiedostossa olevia arvoja, blogi ei saisi yhteyttä tietokantaan eikä käyttäjä voi kirjautua sisään.

7 Kehittämistyön tulokset

Kehittämistyön tuloksena on toimiva blogi. Lopputulos ei täysin täsmää, mitä olin alun perin suunnitellut. Kehittämistyön tekijänä olin jo varhain päättänyt, että jos jokin asia on

järkevämpi tehdä eri tavalla kuin suunnittelin, tekisin niin. Nämä muutokset olivat loppujen lopuksi pieniä ulkonäköön liittyviä. Kaikki suunnitellut toiminnot on tehty.

Mobiilille tarkoitettu navigaatio on hieman eri näköinen kuin suunnittelin. Alkuperäisessä suunnitelmassa hampurilaisnavigaation -ikonin ympärillä on harmaa ympyrä (kuvio 60), mutta muutin sen kuvion 61 näköiseksi. Syynä oli, että animaatio, joka tulee, kun navigaation avaa, näyttää mielestäni paremmalta, kun sivulla on valmiiksi sivun levyinen harmaa palkki.

Kuvio 60: Alkuperäinen suunnitelma mobiililaitteen navigaatioon

Kuvio 61: Kehitetty mobiililaitteen navigaatio

Alkuperäisessä suunnitelmassa blogikirjoituksen kortissa on kohta *read more*, jota painamalla saa avattua blogikirjoituksen luettavaksi. Kehittäessäni sivua tajusin, että on järkevämpää, jos koko korttia voi painaa, joten poistin *read more* -tekstin. Kehitetyssä blogissa on myös vähemmän animaatioita kuin suunnittelin. Mobiililaitteelle hakukentän piti avautua klikkaamalla suurennuslasia, mutta päätin sijoittaa hakukentän koko ajan näkyväksi. Syvää syytä tähän ei ole, mutta animaation tekeminen ei vain tuntunut järkevältä.

8 Kehittämisehdotukset

Kehittämistyön aikana tarkoituksena oli kehittää blogi perustoiminnoilla. Tätä voin jatkokehittää tämän opinnäytetyön jälkeen. Blogin hallintajärjestelmä voisi olla parempi. Esimerkiksi erilaiset ilmoitukset puuttuvat, kun blogikirjoituksen poistaa, muokkaa tai julkaisee. Poiston yhteydessä olisi hyvä, jos käyttäjältä kysytään, halutaanko blogikirjoitus varmasti poistaa.

Blogin tyylittelyyn liittyen voisi katsoa laitteiden koot paremmin. Tabletteja on todella monen eri kokoisia, joten niille tarkoitettu tyyli voitaisiin jakaa pienempiin osiin. Kehittämistyön aikana tyylitiedostot jaettiin kolmeen eri osaan, kännykkä, tietokonenäyttö ja niiden välille laitoin yhden ns. breakpointin.

Tulevaisuudessa olisi myös mukava, jos blogikirjoituksia voisi kommentoida. Tätä varten lukijoilla täytyy olla mahdollisuus luoda omat käyttäjät, mikä on jo isompi jatkoprojekti tähän kehittämistyöhön. Kommenttimahdollisuudella blogikirjoitukset pysyvät mahdollisesti aktiivisemmin lukijan ja kommentoija mielessä. Koska olen itse EU:ssa ja mahdollisesti uudet käyttäjätkin, täytyy minun ottaa huomioon EU:n laatima GDPR, eli tietosuoja-asetus, jonka tarkoituksena on suojata yksittäisen henkilön henkilötietoja.

Blogikirjoituksien kirjoittamisessa täytyy ottaa myös huomioon mahdolliset eettiset ongelmat. Mitä tietoa päätän jakaa tapauksesta? Miten paljon tuon omaa näkökulmaa tai mielipidettä esille? Mitä asioita saatan jättää sanomatta? Miten neutraali minun jakamani sisältö

tutkimuksesta oikeasti on? Mitä jos henkilöt, jotka liittyvät tapaukseen, eivät halua, että tapausta levitetään netissä?

Mielestäni on tärkeää, että tuon kirjoituksissa esille, että kirjoitus on koottu eri lähteistä jopa eri kielillä, joten tapauksessa olevat yksityiskohdat saattavat olla osittain väärin. Näihin eettisiin ongelmiin auttaa myös lukijoiden kommenttimahdollisuus, sillä he saattavat huomata tai tietää asioita, joita minä en edes ajatellut.

9 Oman oppimisen arviointi

Opin todella paljon tämän kehittämistyön aikana. Vaikka idea, miten eri toiminnot kehitetään olivat minulle selkeitä, en ollut aikaisemmin kehittänyt esimerkiksi CRUD -toimintoja enkä kirjatumista, joka vaatii tietokannasta käyttäjän tiedot. Tässä kehittämistyössä opin paljon erilaisten toimintojen käyttöä, joita voin käyttää myös muissa MERN -pinolla tehdyissä projekteissa.

Kehittämisvaiheessa ajattelin useammassa kohdassa, että minun olisi pitänyt suunnitella blogi paremmin. Useammassa kohdassa huomasin ajatelleeni, että en oikein tiedä, mitä seuraavaksi kannattaisi tehdä. Minun olisi pitänyt jakaa koko projekti paremmin pienempiin osiin, jotta olisin voinut keskittyä paremmin sen tekemiseen. Tietenkin tämä oli ensimmäinen kerta, kun tein isomman projektin, joten ajattelen tätä projektia lähtökohtana ja osaan tehdä seuraavassa projektissa paremmin.

Lähteet

Sähköiset

Ashutosh, K. 2021. Progressive web apps – the future of the modern web? Hongkiat. Viitattu 23.3.2022. https://www.hongkiat.com/blog/progressive-web-apps-future-of-modern-web/

Bradford, L. 2021. Web-suunnittelu vs. Web-kehitys: Mikä on ero? Chalized. Viitattu 18.8.2021. https://fi.chalized.com/web-suunnittelu-vs-web-kehitys-mikae-on-ero/

Bisht, S. 2013. Robot framework test automation. E-kirja. Iso-Britannia: Packt Publising.

Codecademy 2021. Back-End Web Architecture. Viitattu 18.10.2021. https://www.codecademy.com/articles/back-end-architecture

Coleman, B. & Goodwin, D. 2017. Designing UX: Prototyping, 1st edition. E-kirja. Australia: SitePoint Pty. Ltd.

Express 2021. Fast, unopiniotaned, minimalist web framework for Node.js. Viitattu 21.10.2021. https://expressjs.com/

Fitzgerald, A. 2021. API Calls: What they are & how to make them in 5 easy steps. Viitattu 4.4.2022. https://blog.hubspot.com/website/api-calls

Foster, E & Godbole, S. 2016. Database Systems, 2nd edition. E-kirja. Yhdysvallat: Merkle Inc.

Friis Dam, R. & Yu Siang, T. 2020. Design thinking: get started with prototyping. Interaction design foundation. Viitattu 21.9.2021. https://www.interaction-design.org/literature/article/design-thinking-get-started-with-prototyping

Gonzales, J. 2013. Mobile first design with HTML5 and CSS3. E-kirja. Iso-Britannia: Packt Publishing Ltd.

Google search central 2021. Learn about sitemaps. Viitattu 27.9.2021. https://developers.google.com/search/docs/advanced/sitemaps/overview

Graham, G. 2015. The difference between responsive and adaptive design. Css-tricks. Viitattu 3.11.2021. https://css-tricks.com/the-difference-between-responsive-and-adaptive-design/

Hietaniemi, J. 2019. Scrum pähkinänkuoressa. Gofore. Viitattu 20.11.2021. https://gofore.com/scrum-pahkinankuoressa/

Hietaniemi, J. 2020. Mikä on Kanban? Gofore. Viitattu 20.11.2021. https://gofore.com/mika-on-kanban/

Hoque S. 2018. Full-stack reactprojects: modern web development using React16, Node, Express, and MongoDb. E-kirja. Iso-Britannia: Packt Publishing Ltd.

Hoque S. 2020. Full-stack react projects: learn MERN stack development by building modern web apps using MongoDB, Express, React and Node.js. E-kirja. Iso-Britannia: Packt Publishing Ltd.

Hughes, J. 2019. Web apps vs websites: What's the difference? Does it matter? Themeisle. Viitattu 5.8.2021. https://themeisle.com/blog/web-apps-vs-websites/

Json 2021. Introducing JSON. Viitattu 21.10.2021. https://www.json.org/json-en.html

Koulutus.fi 2021. Mitä ovat ketterät menetelmät? – Scrum, Lean ja muut tutuksi. Viitattu 15.11.2021. https://www.koulutus.fi/oppaat/projektinhallinta/ketteratmenetelmat-19939

Leppäniemi, P. 2021. Avoin lähdekoodi: mitä ja miksi. Fraktio. Viitattu 2.11.2021. https://www.fraktio.fi/blogi/avoin-lahdekoodi-mita-ja-miksi

Letendart 2018. What is web development? Openclassrooms. Viitattu 18.8.2021. https://blog.openclassrooms.com/en/2018/03/28/web-development-definition/

Lopuck L. 2021. Web design for dummies. 3.painos. E-kirja. New Jersey: John Wiley & Sons, Inc.

Markdown.fi 2022. Markdown. Viitattu 4.4.2022. https://www.markdown.fi/

Mead, A. 2018. Learning Node.js development: learn the fundamentals of Node.js, and deploye and test Node.js application on the web. E-kirja. Iso-Britannia: Packt Publishing Ltd.

MongoDB 2021. MERN stack. Viitattu 23.9.2021. https://www.mongodb.com/mern-stack

MongoDB 2021. What is an object-oriented database? Viitattu 19.10.2021. https://www.mongodb.com/databases/what-is-an-object-oriented-database

Neal, D. 2020. The best testing tools for node.js. Viitattu 10.11.2021. https://developer.okta.com/blog/2020/01/27/best-nodejs-testing-tools

Oracle 2021. What is a relational database(RDBMS)? Viitattu 19.10.2021. https://www.oracle.com/database/what-is-a-relational-database/

Pixels 2021. Web-sovellukset. Viitattu 5.8.2021. https://pixels.fi/fi/web-sovellukset/

React 2021. React. Viitattu 25.10.2021. https://reactjs.org/

Redandblue 2021. Web sovellus tuo natiisovelluksen ominaisuudet verkkopalveluun. Viitattu 5.8.2021. https://redandblue.fi/fi/web-sovellukset/

Shinobi 2021. What is a sitemap? Exposure ninja. Viitattu 28.9.2021. https://exposureninja.com/training/guides/seo/what-is-a-sitemap/

StaffMill 2021. Mikä on Full-stack developer? Viitattu 27.9.2021. https://www.staffmill.fi/nl/mika-on-full-stack-developer/

UXPin 2021a. Responsve Design vs. Adaptive Design: what's the best choice for designers? Viitattu 2.11.2021. https://www.uxpin.com/studio/blog/responsive-vs-adaptive-design-whats-best-choice-designers/

UXPin 2021b. A hands-on guide to mobile-first responsive design. Viitattu 22.11.2021. https://www.uxpin.com/studio/blog/a-hands-on-guide-to-mobile-first-design/

Yrityksen perustaminen.net 2021. Laadukkaat verkkosivut pienelle yritykselle – miksi, mistä ja miten? Viitattu 26.8.2021. https://yrityksen-perustaminen.net/kotisivut-yritykselle/

Kuviot