Merili Elnadi

Pussisuodattimien energialaskuri web-sovelluksena

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

28.5.2024

Tiivistelmä

Tekijä:	Merili Elnadi
Otsikko:	Pussisuodattimien energialaskuri web-sovelluksena
Sivumäärä:	27 sivua
Aika:	28.5.2024

Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Mobile Solutions
Ohjaajat:	Toni Spännäri

Insinöörityön tarkoituksena oli päivittää toimeksiantajan nykyinen pussisuodattimien energialaskuri web-sovellukseksi. Työn keskiössä oli laskurin ja sen helppokäyttöisyyden sekä tietoturvan parantaminen. Uusi versio loi myös kehitysmahdollisuuksia uusille ominaisuuksille.

Uuden web-sovellusversion suunnittelussa ja toteutuksessa kiinnitettiin erityistä huomiota alkuperäisen toiminnallisuuden säilyttämiseen, yrityksen brändi-ilmeen integroimiseen ja vakaan pohjan luomiseen sovelluksen jatkokehitystä ajatellen. Web-sovellus toteutettiin käyttäen JavaScript-ohjelmointikieltä, Node.js-kehystä ja tietokantaratkaisuna MariaDB:n ympäristöä. Projektin aikana ratkaistiin tehokkaasti erilaisia teknisiä haasteita liittyen muun muassa tietokannan integrointiin ja laskurin laskelmien toteuttamiseen.

Insinöörityön lopputuloksena syntyi toimiva laskuri pohjaksi varsinaiselle lopputoteutukselle, joka on tulevaisuudessa osana toimeksiantajan asiakasportaalia. Uuden web-sovelluksen kehitys ja käyttöönotto oli merkittävä askel kohti asiakaslähtöisempää palvelua tarjoten tehokkaamman tiedonkeruun ja -hallinnan verrattuna edelliseen toteutukseen.

Avainsanat: web-sovelluskehitys, tietoturva, Node.js

Abstract

Author:	Merili Elnadi
Title:	Pocket Filter Energy Calculator as a Web Application
Number of Pages:	27 pages
Date:	28.5.2024
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Professional Major:	Mobile Solutions
Supervisors:	Toni Spännäri, Senior Lecturer

The objective of the study was to update the client's current pocket filter energy calculator into a web application. The focus of the development work was on improving the usability and security of the existing calculator. The new version created new opportunities for new features.

In the design and implementation of the new application version, special attention was paid to maintaining the original functionality, integrating the brand identity of the company, and creating a stable foundation for the future development of the application. The web application was implemented using the JavaScript programming language, the Node.js framework and the MariaDB environment as the database solution. Various technical challenges related to the database integration and the implementation of the performed calculations were effectively addressed during the project.

As a result of the study, a functional calculator was developed as a starting point for the actual implementation, which will be part of the client's customer portal in the future. The development and deployment of the new web application was a significant step towards a more customer-centric service, offering more efficient data collection and management compared to the previous solution.

Keywords: web application development, information security, Node.js

Sisällys

Lyhenteet

HTML: *Hyper Markup Language.* Standardoitu merkintäkieli, jolla määritellään verkkosivujen rakenne ja sisältö.

CSS: *Cascading Style Sheets*. Tyylittelykieli, jolla muokataan verkkosivun ulkoasua ja responsiivisuutta.

HTTP: *Hypertext Transfer Protocol.* Verkkoympäristöissä tiedonsiirtoon liittyvä protokolla.

JSON: *JavaScript Object Notation.* Tiedonvälitykseen ja tallennukseen käytetty tietomuoto.

1 Johdanto

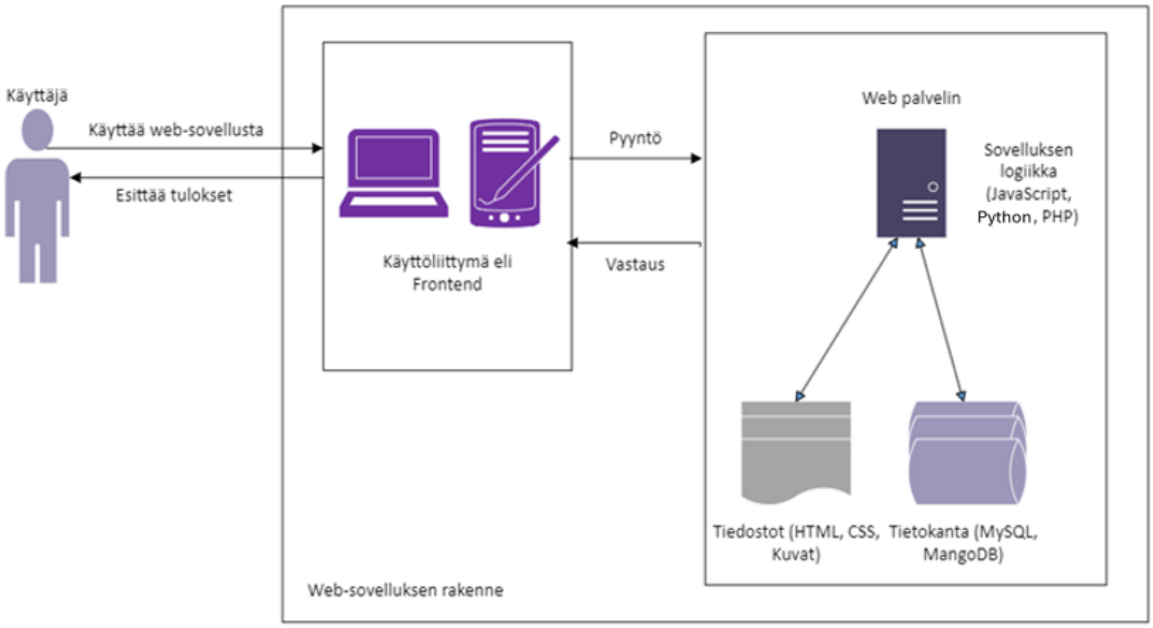
Tämän insinöörityön tarkoituksena on päivittää Dinair Clean Air Oy:n pussisuodattimien energialaskuri web-sovelluksen muotoon. Nykyisen Excel-pohjaisen laskurin tilalle toivotaan helppokäyttöisempää versiota. Insinöörityössä paneudutaan tarkasti laskurin päätoimintoihin ja dokumentoidaan uuden version kehitysprosessi. Työssä on erityisen tärkeää ottaa huomioon yrityksen vaatimukset ja toiveet sekä ymmärtää olemassa olevat toimintatavat. Yksinkertaisen ja helppokäyttöisen laskurin toivotaan olevan arvokas työkalu myyntitilanteissa ja selkeyttävä tekijä asiakkaan päätöksenteossa.

Insinöörityön toimeksiantajana toimiva Dinair Clean Air Oy on toiminut vastuullisena ilmansuodattimien valmistajana jo vuodesta 1979. Dinair Clean Air Oy on osana AAF International-konsernia, joka on maailman suurin ilmansuodatusratkaisujen valmistaja. AAF International on osana emoyhtiötä Daikinia. Suomessa Dinair Clean Air Oy on erikoistunut erityisesti pussi- ja paneelisuodattimiin, jotka ovat olennainen osa ilmanpuhdistusjärjestelmiä erilaisissa tiloissa, kuten kouluissa, sairaaloissa, toimistorakennuksissa ja ostoskeskuksissa. Tämä yritys arvostaa suuresti kotimaista tuotantoa, erinomaista asiakaskokemusta ja ympäristöystävällistä toimintaa.

Dinair Clean Air Oy:n tavoite vuoteen 2025 mennessä on puolittaa pussisuodattimien hiilijalanjälki. Tähän on jo panostettu vähentämällä valmistuksessa syntyvää jätettä ja optimoimalla materiaalien käyttöä. Lisäksi yhteistyökumppanit on valittu ajatellen hiilijalanjäljen pienentämistä. Yrityksessä ohjataan ja tuetaan myös asiakkaita tekemään ympäristöystävällisiä valintoja laajasta tuotevalikoimasta. Tämän insinöörityön tuloksena syntyvä verkkopohjainen energialaskuri tukee edellä mainittua toimintatapaa.

2 Modernin web-sovelluksen rakenne

Web-sovellukseksi kutsutaan sovellusta, jota voidaan käyttää erilaisilla verkkoselaimilla laitteesta riippumatta. Perinteisen web-sovelluksen rakenne koostuu kolmesta osasta, jotka ovat frontend (selainpuoli), backend (palvelinpuoli) ja tietokanta. (Volle 2022.) Frontend toimii käyttäjälle käyttöliittymänä ja backend hoitaa tarvittavat prosessit taustalla. Nämä prosessit ovat esimerkiksi käyttäjän pyynnöstä tehdyt lomakkeiden lähetykset ja tietokantakyselyt vastauksena kyseisiin pyyntöihin. Kaikkien näiden osien välinen vuorovaikutus mahdollistaa sovelluksen toiminnallisuuden ja käyttökokemuksen loppukäyttäjälle. Kuvassa 1 on esitetty tämä jako.



Kuva 1. Web-sovelluksen rakenne (Mukaillen: Dabbs 2019).

Web-sovelluksen erottaa perinteisestä verkkosivusta toiminnallisuuden perusteella. Verkkosivu vain esittää tietoa toisin kuin web-sovellus, joka suorittaa myös erilaisia toimintoja. (Craig; Pollard & Strum 2017.) Nämä toiminnot voisivat esimerkiksi olla hakutoiminnot, viestittelytoiminnot tai yhteys verkkopankkiin maksamista varten.

2.1 Frontend

Frontend eli käyttöliittymä koostuu usein HTML-, CSS- ja JavaScript-kielistä, jotka yhdessä määrittävät web-sovelluksen ulkoasun. HTML on merkintäkieli, jonka avulla määritellään verkkosivun rakenne ja elementit, kuten esimerkiksi otsikot, kappaleet ja linkit. HTML:ää käyttämällä luodaan sivun perusrakenne, jota CSS ja JavaScript muokkaavat sekä täydentävät.

CSS-kieli muokkaa web-sivun ulkoasua ja visuaalista tyyliä muuttaamalla HTML-elementtien ulkonäköä. Kielen avulla onnistuu esimerkiksi taustojen ja fonttien muokkaus. Lisäksi CSS mahdollistaa sivustojen muokkaamisen sopivaksi erikokoisille näytöille. CSS tarjoaa keinoja muun muassa tyylisääntöjen hallintaan ja organisointiin. Esimerkiksi tyyliluokkien luominen CSS:n avulla mahdollistaa tiettyjen tyylittelysääntöjen soveltamisen valittuihin HTML-elementteihin tai -ryhmiin. (Abramowski 2023.)

Web-sovelluksen frontendin toteuttamisessa kuitenkin usein käytetään jotakin JavaScript-kirjastoa tai -sovelluskehystä peruskielien lisäksi. Tunnetut ja yleisesti käytössä olevat kirjastot React ja Angular nopeuttavat sovelluskehittäjän työtä tarjoamalla valmiita rakenteita ja komponentteja sekä kasvattavat sovelluskehitysyhteisön yhteisöllisyyttä. Esimerkiksi Reactilla toteutettuja komponentteja voidaan jakaa eteenpäin muille kehittäjille, mikä tehostaa kehitystyötä. Tässä yhteydessä komponentti viittaa siis kokonaisuuteen, joka koostuu yhdestä tai useammasta HTML-elementistä. Esimerkkikoodissa 1 on esitetty komponentti, joka on toteutettu React-kirjastoa käyttäen.

```
function Welcome(props){
    return <h1>Hello, {props.name}</h1>;
}
const element = <Welcome name="Merili Testi" />;
ReactDOM.render(
    element,
    document.getElementById('root')
);
```

Esimerkkikoodi 1. React-elementin esimerkki.

2.2 Backend

Yleisimmät ohjelmointikielet backendissa ovat JavaScript, Python, Java, PHP ja Kotlin. Kuten sanottu backend eli palvelinpuoli toimii sovelluksen ytimenä ja muodostaa toiminnallisen perustan. Logiikka prosesseihin, tehtäviin ja tiedonhallintaan tulee juuri backendin kautta. Useissa lähteissä myös tietokanta määritellään osaksi backendia. Todella yksinkertaistetusti backend välittää ja käsittelee tietoja käyttöliittymän ja tietokannan välillä. Tyypillisiä backendissa tapahtuvia toimintoja ovat esimerkiksi sisäänkirjautuminen ja käyttäjähallinta (autentikointi), tietokantakäsittely sekä virheiden käsittely ja lokitiedostojen hallinta.

Myös backendin ohjelmointiin on kehitetty useita kehyksiä ohjelmistokehityksen mahdollisuuksien llaajentamiseksi ja työn helpottamiseksi. Kehykset tarjoavat muun muassa valmiita moduuleja ja komponentteja toimintojen toteuttamiseen sekä työkaluja tietokantakäsittelyyn. Näitä kehyksiä ovat muun muassa Express.js, Django, Spring Boot ja Laravel. Jokainen mainituista toimii erilaisella ohjelmointikielellä. (Tomar 2023.) Express.js tarjoaa perusominaisuuksia, kuten reittien ja väliohjelmistojen hallintaa sekä HTTP-pyyntöjen käsittelyä. Express.js toimii JavaScript-kielellä Node.js ympäristön päällä. Myös muut mainitut

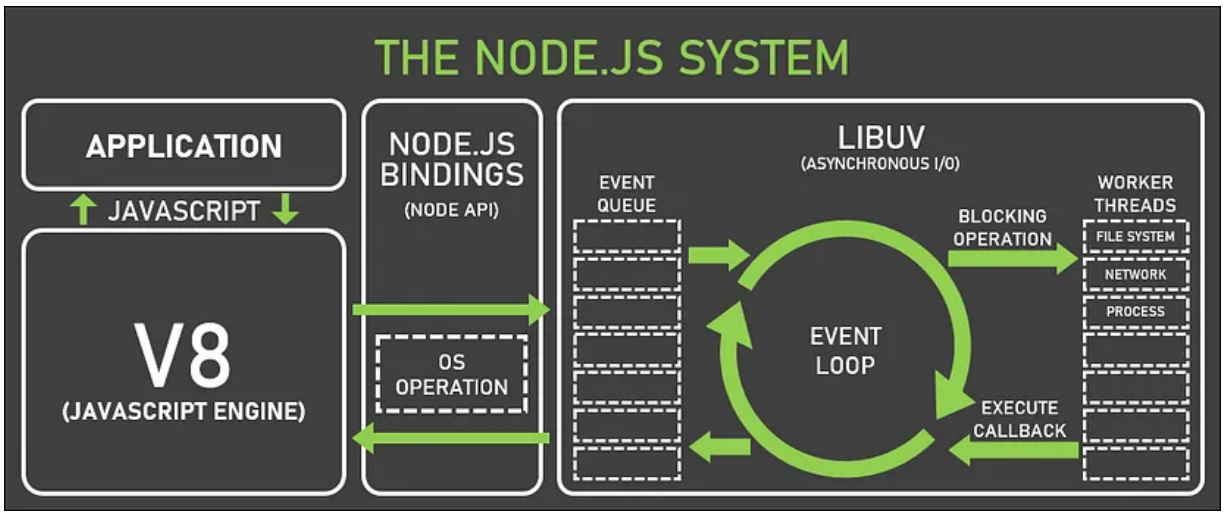
kehityskehykset tarjoavat runsaasti valmiita ominaisuuksia, työkaluja ja rakenteita ohjelmistokehitykseen. Sopiva kehys ja kirjastot täytyy valita projektin laajuuden, sovelluksen käyttötarkoituksen ja kehittäjän osaamisalueen mukaan.

Node.js

Tässä työssä käytetty Node.js on avoimen lähdekoodin suoritusympäristö JavaScript-koodin suorittamiseen palvelimen puolella. Ympäristö on julkaistu vuonna 2013, ja sitä kehitetään jatkuvasti. Tällä hetkellä viimeisin versio on julkaistu 8.3.2024. (About Node.js 2024.) Tämä on valittu käyttöön kyseiseen projektiin aikaisemman kokemuksen takia. Tarkoitus on edistää olemassa olevaa osaamista lisää.

Node.js on suosittu sovelluskehittäjien keskuudessa. Se tarjoaa tehokkaan, nopean ja joustavan ympäristön palvelinsovelluskehitykselle JavaScriptillä. Node.js nimittäin tukee paketteja hallinnoivaa npm (Node Package Manager)-työkalua, jonka avulla kirjastojen ja riippuvuuksien asennus sekä päivittäminen onnistuvat helposti [An introduction to the npm package manager 2024].

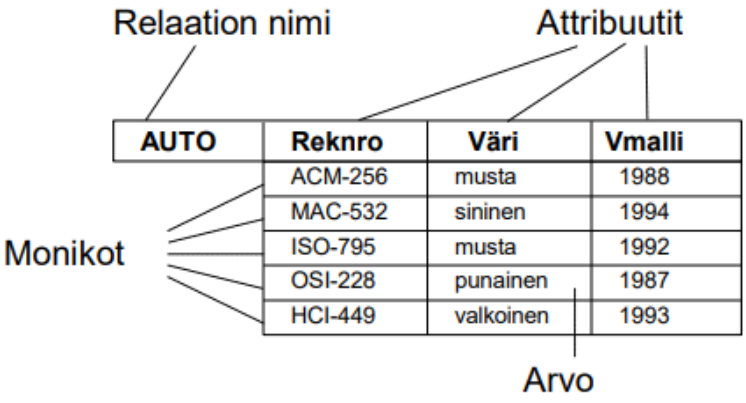
Kuvassa 2 on esitetty Node.js:n toimintalogiikkaa hieman tarkemmin. Ytimenä toimii event loop eli tapahtumasilmukka, joka vastaanottaa ja käsittelee tapahtumia yksisäikeisesti ja asynkronisesti. Asynkronisuus tarkoittaa sitä, että ohjelman suoritus ei pysähdy odottamaan pyynnön vastausta. Sen sijaan se voi siirtyä seuraavien pyyntöjen suorittamiseen, minkä jälkeen voidaan palata ensimmäisen pyynnön vastauksen käsittelyyn. Node.js:n tehokkuus perustuu siis yhden säikeen tapahtumasilmukkaan ja JavaScriptin ominaisuuksiin, kuten ensimmäisen ja korkeamman asteen funktioihin. Nämä funkiotyypit mahdollistavat funktion asettamisen parametriksi toiseen funktioon.



Kuva 2. Node.js:n toimintalogiikka (Nagarajan 2019).

2.3 Tietokanta

Tietokannat ovat organisoituja tiedon kokoelmia, jotka sisältävät taulukoita ja tietorivejä. Usein taulukot liittyvät toisiinsa. Tässä työssä käytetty relaatiotietokannan hallintajärjestelmä MariaDB käyttää rakennekyselyitä (SQL-kyselyt) tietokantojen toimintojen suorittamiseen. (Opi MySQL/MariaDB aloittelijoille - Osa 1.) Relaatiomallisen tietokannan tietueiden relaatiologiikka on esitetty kuvassa 3, josta nähdään, että esimerkkitaulukossa tiettyä autoa vastaavat tietty väri ja vuosiluku.



Kuva 3. Relaatiomallisen tietokannan logiikka (Laine 1999).

Hyvässä tietokannassa täytyy esitellä tiedot loogisesti ja ymmärrettävästi eikä tietokannassa pitäisi olla päällekkäisyyksiä. Tietokannan ylläpitoa ajatellen on tärkeää, että taulujen käyttöoikeudet, varmuuskopiointi ja dokumentointi ovat kunnossa sekä ajan tasalla.

Murhpy [2024] listaa artikkelissaan tietokantoihin liittyvät yleiset uhat. Näitä ovat yrityksen sisäiset riskit, jossa esimerkiksi työntekijät muuttavat, poistavat tai lähettävät edelleen salaista tietoa. Lisäksi heikon salasanan käyttö tai saman salasanan käyttäminen pitkään voi olla uhkana. Artikkelissa mainitaan uhkina myös hallintajärjestelmän heikkoudet, järjestelmävirheet tai hakkerihyökkäykset, joissa käytetään vaarallisia SQL-injektioita. Kyseisten ongelmien välttämiseksi on tärkeää pitää huoli tietokannan palomuurista, enkryptauksesta ja käyttöoikeuksien sekä salasanojen ylläpidosta. (Murphy 2024.) On myös erittäin hyödyllistä pitää lokikirjaa tietokantaan tehdyistä muutoksista ja lukukerroista käyttäjäkohtaisesti jäljitettävyyden takia.

2.4 Kommunikaatio

Backendin, frontendin ja tietokannan väliseen liikenteeseen ja kommunikaatioon käytetään yleisesti Rest API -rajapintaa. Tämän rajapinnan käyttö ei vaadi erillistä kirjastoa, ja se voi käsitellä monenlaisia pyyntöjä. Tieto siirretään usein JSON- tai XML-tallennusmuodossa, jota voi muokata tarpeen mukaiseksi. Reaaliaikaiseen tiedonsiirtoon tämä vaihtoehto ei kuitenkaan sovellu. (Halili & Ramadani 2018.) Rest API -rajapintaan liittyvät HTTP-pyynnöt siirtävät dataa muun muassa seuraavien metodien mukaisesti:

- GET-pyynnöllä haetaan dataa. Käyttäjä suorittaa esimerkiksi hakupalkista haku-toiminnon. Tämä johtaa GET-pyynnön toteutumiseen.

- POST-pyyntö lisää uutta dataa tietokantaan. Tämä tapahtuu esimerkiksi, kun käyttäjä tekee myynti-ilmoituksen sivustolle. Hänen tekemänsä julkaisu ja sen tiedot tallennetaan kyseisellä pyynnöllä.

- PUT-pyyntöä käytetään tiedon korvaamiseen, muokkaamiseen ja päivittämiseen.

- DELETE-pyyntö poistaa datan tietokannasta.

Frontendin ja backendin välinen kommunikaatio voidaan toteuttaa myös esimerkiksi GraphQL:llä tai WebSocketilla. GraphQL tarjoaa joustavuutta ja tehokkuutta, sillä käyttäjä voi määrittää tarkalleen, mitä tietoa hän haluaa saada vastaukseksi pyyntöönsä. Ylimääräisen datan lähettämistä GraphQL:llä tapahtuu huomattavasti vähemmän, mikä mahdollistaa monimutkaiset tietopyynnöt ja useiden eri laitteiden tukemisen samanaikaisesti rajapinnalla. WebSocket on yleensä käytössä sovelluksissa, johon vaaditaan reaaliaikaista, kaksisuuntaista tietoliikennettä. Tämä protokolla mahdollistaa jatkuvan yhteyden palvelimen ja käyttäjän välillä.

Tiedonsiirtoon löytyy myös monia muita vaihtoehtoja, mutta tämän insinöörityön puitteissa on mainittu yleisimmät. Sopiva tapa valitaan sovelluksen tyypin ja käyttötarkoituksen mukaan.

3 Tietoturva

Tietoturvan puute on kasvava huolenaihe nykymaailmassa sekä yritysten että henkilökäytön tasolla. Riskinä ovat esimerkiksi pahantahtoiset toimijat, jotka pyrkivät hyväksikäyttämään sovellusten haavoittuvuuksia. Tavoitteena voi olla vahingon aiheuttaminen tai hyötyminen esimerkiksi salaisen tiedon tai rahan avulla. Hakkerit kehittyvät vähintäänkin samaa tahtia kuin sovelluskehitys yleisesti.

Kun web-sovellusta rakentaa yritykselle, täytyy olla erittäin varovainen tietoturvankin osalta. Sovellus voi käsitellä salaista tietoa, jonka joutuessa vääriin käsiin yritykselle voisi aiheutua liikesalaisuuksien vuotamista, maineen ja luottamuksen vahingoittumista ja taloudellisia menetyksiä. Kuvassa 4 on esitetty pääteemat, jotka täytyy muistaa tietoturvallisen verkkosivun rakentamisessa.

Kuva 4. Tärkeimmät osa-alueet turvallisen verkkokaupan rakentamisessa (N-iX 2020).

Vaikka kuvassa on keskitytty verkkokaupan luomiseen, samat osa-alueet pätevät myös hyvin yleisesti web-kehityksessä. Seuraavaksi käsitellään tapoja, joilla tietoturva täytyy ottaa esimerkiksi tässä sovelluskehitystyössä huomioon.

3.1 Autentikaatio

Käyttäjätilin varmistamisella eli autentikaatiolla varmistetaan, että käyttäjä on aidosti oikea käyttäjä eikä ainoastaan esitä olevansa. Autentikaatiota käytettäessä sovelluskehityksessä on viisasta käyttää olemassa olevaa ja testattua kehystä, joka vastaa sovelluksen tarpeita. Yleistä käyttäjätunnus-salasana-yhdistelmää ei pidetä enää kovin turvallisena ainoana kirjautumisen menetelmänä, sillä käyttäjät käyttävät usein samaa salasanaa monissa paikoissa ja monet salasanat ovat helposti arvattavissa tai selvitettävissä. (Identification and Authentication Failures, 2021.)

Perinteisen käyttäjätunnuksen ja salasanan autentikaation sijaan tai rinnalla käytetään esimerkiksi biometrisia autentikaation keinoja tai Single sign-on (SSO) -menetelmää. Biometrinen autentikaatio vaatii tietyn sormenjäljen tai kasvonpiirteet käyttäjätilin varmistamiseksi. SSO-menetelmässä käyttäjä kirjautuu vain kerran, mutta pääsee käyttämään erilaisia sovelluksia, jotka on kytketty toisiinsa. Vaikka tämä tapa ei kuulosta olevan kovin turvallinen, siitä löytyy monia hyviä puolia, kun tarkastelee tavallisen käyttäjän verkkokäyttäytymistä ja tottumuksia. Tämän tavan hyödyt ovat esimerkiksi kirjautumisten keskittäminen, monipuoliset salasanat ja helppo sekä selkeä tarkastus ja seuranta. (Agarwal.) Kun käyttäjän täytyy muistaa vain yksi salasana, hän todennäköisemmin valitsee vaikeamman tai pidemmän salasanan verrattuna tilanteeseen, jossa hänen täytyisi muistaa monia salasanoja kerralla. SSO-autentikaatio on käytössä isolla alustoilla, joita ovat muun muassa Salesforce, Microsoft Office 365 ja Facebook.

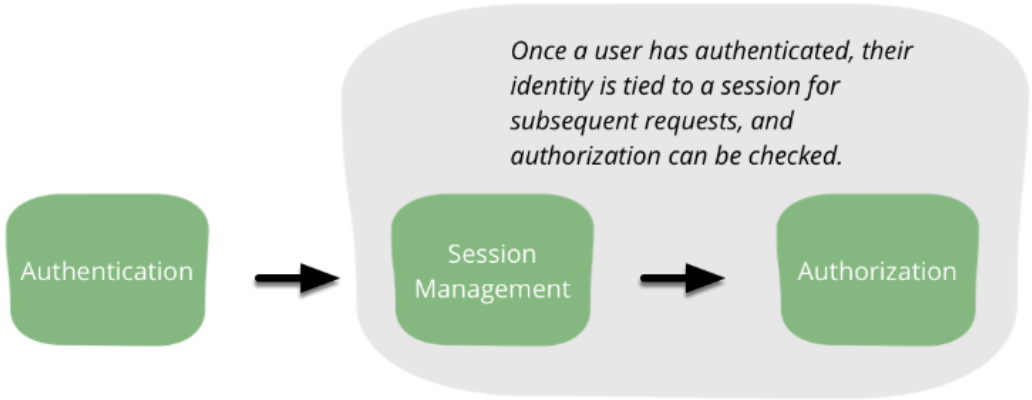
Salasanojen hallinnassa on tärkeää muistaa myös salasanojen voimassaoloajat ja tallennusmenetelmät. Sovelluskehittäjän näkökulmasta on hyvä muistaa, että salasanoja ei ikinä kannattaisi tallentaa sellaisenaan tietokantaan. Salasanat täytyy vähintäänkin hajauttaa (engl. hashing), jolloin algoritmit muuttavat salasanat muuttumattomiksi kryptisiksi arvoiksi. Tämän lisäksi käytetään usein myös salt-menetelmää, jossa yhdistetään satunnaisesti generoitu arvo osaksi hash-arvoa. Tämä tapa yksilöi käyttäjien salasanat, mikä tekee niiden arvaamisesta hankalan hyökkääjille. (Password Hashing and Salting Explained 2023.) Vaikka esimerkiksi kahden käyttäjän salasanat ovat samat, tietokannassa nämä salasanoihin viittaavat arvot ovat kuitenkin erilaiset. Tallennustapojen lisäksi on tärkeä asettaa salasanoille voimassaoloajat, jotta mahdollisuus pitkäaikaiselle salasanan väärinkäytölle olisi mahdollisimman pieni.

3.2 Käyttöikeudet

Valtuutus (authorization) viittaa käytäntöön, jossa käyttäjän istunnon aikana ohjelma tarkistaa, riittävätkö käyttäjän oikeudet tietyille toiminnoille tai web-sivun käytölle yleisesti. Web-sovellusta rakennettaessa on siis tärkeä ottaa

huomioon käyttöoikeuksiin liittyvät haavoittuvuudet, joissa käyttäjä pääsee käsiksi tietoihin, joihin hänellä ei pitäisi olla pääsyä. Lisäksi ovat ongelmallisia haavoittuvuudet, joissa pystytään manipuloimaan käyttöoikeuksia tai kiertämään ne täysin. Tällainen haavoittuvuus sovelluksessa voi olla hyvinkin laaja ja edetä ajan kanssa. Tämä tarkoittaa, että hyökkääjä voi esimerkiksi päästä käyttämään järjestelmävalvojan työkaluja ja ominaisuuksia, mikä voi aiheuttaa entistä enemmän tuhoa.

Session Management eli suomeksi istunnon hallinta on myös erittäin tärkeä osa web-sovelluksen tietoturvaa. Istunnon hallinta ylläpitää hyvää käyttäjäkokemusta. Käyttäjän toiminnot ja muutokset tallentuvat ilman, että hänen täytyy käytön aikana uudelleenkirjautua. Käytettävyyden lisäksi tämä myös estää hyökkääjiä ottamasta hallintaansa kyseistä istuntoa eli käyttökertaa. Usein istunnon tunniste tallennetaan evästeeseen tai muihin vastaaviin mekanismeihin, jolloin tunnisteen selvittäminen on hankalaa. Istunnon hallinnassa tapahtuva istunnon aikakatkaisu ja automaattinen uloskirjautuminen auttavat tilin ja tietojen suojaamisessa erityisesti, kun käyttäjä on ollut kirjautuneena julkisella tai jaetulla laitteella.



Kuva 5. Web-sovelluksen autentikaatio, istunnon hallinta ja valtuututtaminen (Cairns & Somerfield 2017).

Kuvassa 5 autentikaatio, istunnon hallinta ja valtuututtaminen on esitetty selkeästi ja loogisesti. Yksinkertaistetusti autentikaation jälkeen käynnistyy

istunnon hallinta, jonka aikana sovellus antaa pääsyn käyttäjälle tarkoitettuihin näkymiin tai toimintoihin. Pääsyn antaminen eli valtuutuuttaminen (authorization) toimii kirjautumistietojen ja istunnon hallinnasta saadun tunnisteen perusteella.

3.3 Syötteen validointi

Cairns ja Somerfield (2017) mainitsevat tekstissään HTML-lomakkeen syöttöelementin olevan mahdollinen uhka web-sovelluksissa. Etukäteen ei voida ikinä olla varmoja, minkälaista dataa käyttäjä lomakkeeseen syöttää. Input validation eli syötteen tarkistus on siis oleellinen tekijä tämän uhan käsittelemisessä. Ohjelman täytyy tarkistaa joillain koodatuilla säännöillä vastaako käyttäjän syöttämä tieto odotettua sisältöä tai muotoa. Lisäksi käyttäjälle saapuva tieto virheellisestä syötetystä tiedosta täytyisi olla mahdollisimman vähäistä ja neutraalia, jottei mahdollinen hyökkääjä voisi siitä tiedosta hyötyä. (Cairns & Somerfield 2017.)

Yleisesti sovellusta rakennettaessa kehittäjän täytyy pitää mielessä tietoturvakysymykset jo suunnitteluvaiheessa. Jos haluaa varmistua tietoturvallisuudesta, on parasta käyttää valmiina olevia, testattuja menetelmiä ja kehyksiä, ennen kuin rakentaa niitä täysin itse ilman asiantuntemusta . Tietoturvaan täytyy olla monikerroksinen lähestymistapa, jossa otetaan huomioon mahdollisimman moni uhka ja käytettävyys. Syötteen validoinnin, autentikaation ja käyttöoikeuksien hallinnan lisäksi täytyy muistaa salata verkko- ja tietokantaliikenne esimerkiksi HTTPS-protokollan avulla, pitää ohjelmistoversiot ja riippuvuudet ajan tasalla sekä tehdä koodiarviointeja ja tietoturvatestausta.

4 Nykyinen versio ja suunnittelu

Energialaskuria käytetään myyntitilanteissa ja tuotekehityksessä. Laskuri on merkittävä työkalu, jolla luodaan helposti ymmärrettäviä ja selkeitä laskelmia sekä vertailuja myytävistä pussisuodattimista. Tämä antaa asiakkaille kokonaisvaltaisemman kuvan pussisuodattimien valikoimasta, kustannuksista ja energiakulutuksesta . Seuraavissa luvuissa perehdytään nykyiseen

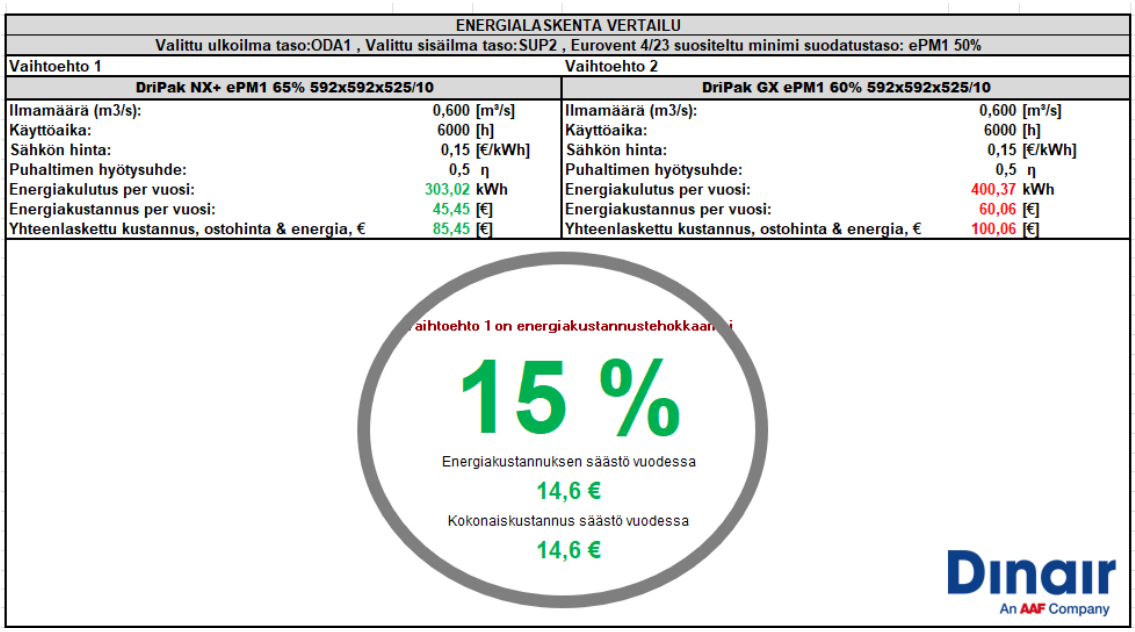
energialaskuriin, sen toiminnallisuuksiin ja mahdollisiin puutteisiin sekä uuden version suunnitteluvaiheeseen.

4.1 Nykyinen versio

Tällä hetkellä käytössä oleva pussisuodattimien energialaskuri toimii Excel-pohjaisena. Excel-tiedostossa on useita välilehtiä, jotka sisältävät muun muassa laskelmiin käytetyt tiedot, käyttöohjeet, termien käännökset ja tulossivun, joka usein jaetaan eteenpäin. Laskurin pääsivu, johon syötetään arvoja laskelmia varten, on helppokäyttöinen ja selkeä toisin kuin tiedosto kokonaisuudessaan.

Käyttäjän tulee syöttää 10 erilaista arvoa, jotta voidaan vertailla kahta erilaista tapausta. Näitä arvoja ovat esimerkiksi pussisuodattimen leveys, korkeus, suodatinmateriaali ja energian kustannukset. Useimmiten vertaillaan pussisuodattimien mallien eroja, mutta on mahdollista myös tarkastella esimerkiksi ilmamäärän tai käyttötuntien vaikutusta energiakulutukseen ja kokonaiskustannuksiin. Kun laskuriin syötetään oikeat tiedot, se käyttää kaavoja ja testidataa laskiakseen molempien pussisuodatinmallien energiakulutuksen ja -kustannukset vuositasolla. Tuloksena laskuri ilmoittaa energiakustannustehokkaamman vaihtoehdon kahdesta.

Kuvassa 6 on esitetty esimerkkilaskelman tulokset tulossivulla. Kyseinen sivu oikeilla laskelmilla esitetään asiakkaalle PDF-muodossa tai kuvakaappauksena. Sivulla on värikoodeilla tuotu esiin energiakustannustehokkaampi vaihtoehto kahdesta ja tulosarvot energiakustannukselle ja energiakulutukselle vuositasolla. Koska kaikki vertailtavat arvot ja mallinimet ovat vierekkäin, asiakkaan on helppo nähdä erot kahden vaihtoehdon välillä. Vertailukontekstissa värikoodien käyttö on yleensä erittäin tehokas keino ohjata käyttäjän huomiota.



Kuva 6. Tulossivu nykyisestä energialaskurista.

Nykyinen laskuri on kokonaisuudessaan Excel-tiedostossa. Vaikka laskurille onkin asetettu salasana, sen käytössä piilee tietoturvauhkia. Kuten on jo todettu, käyttöoikeuksien hallinnointi on tärkeää, jotta sovellus olisi tietoturvallinen. Tämä pätee myös nykyiseen toteutukseen. Excel-tiedosto sisältää kaikki laskukaavat, lähtödatan ja muut liikesalaisuudet, joiden pitää olla vain työntekijöiden käytössä ja tiedossa. Koska laskuri on Excel-tiedostossa, sitä on varsin helppo jakaa eteenpäin täysin ulkopuolisille. Nämä ovat syitä muiden joukossa, miksi laskuria on tarkoitus päivittää.

Koska Excel-tiedostossa on paljon nähtävissä ja muokattavissa olevaa dataa, se altistuu riskille tiedon muuttamisesta joko tahattomasti tai tahallisesti. Tämä voi johtaa laskelmien virheelliseen lopputulokseen, mikäli lähtödataa muutetaan tai laskukaavoja muokataan. Lisäksi mahdollisuus nähdä kaikki välilehdet voi olla uusille käyttäjille hämmentävää.

Puutteellinen tietoturva estää tällä hetkellä laskurin suoran käytön asiakkailta. Uuden version myötä suora käyttö voisi toteutua, sillä sovelluksella voisi olla autentikaatiojärjestelmä, jossa vaaditaan käyttäjän sisäänkirjautuminen. Uusi energialaskurisovellus voisi tulevaisuudessa olla osana yrityksen asiakasportaalia, jolloin asiakkaat pääsisivät tekemään vertailuja portaalista

esimerkiksi ostoprosessin yhteydessä. Tämä parantaisi käyttökokemusta ja antaisi asiakkaille välittömän tiedon energiakustannustehokkaista tuotteista. Tämä on myös yksi pääsyistä, minkä takia uutta versiota on toivottu ja aloitettu työstämään. Excel-laskurin yhdistäminen osaksi asiakasportaalia ja verkkokauppaa olisi vaikeaa ja jatkokehtiys olisi erittäin hankalaa.

4.2 Suunnittelu

Uuden version toiminnallisuuksien ja ulkoasun suunnittelulle on varsin tiukat rajoitukset, sillä laskurin perustoiminnot pysyvät samoina ja ulkoasun täytyy vastata yrityksen vaatimuksia. Vaatimukset edellyttävät huolellista suunnittelua, jotta lopputuloksena syntyvä sovellus säilyttää käytettävyyden ja sopeutuu yrityksen brändi-ilmeeseen. Myös suunnitteluvaiheessa on hyvä ottaa huomioon mahdolliset tulevaisuuden kehitysmahdollisuudet . Tällä tavalla varmistetaan, että uusi versio ei ainoastaan vastaa nykyisiä tarpeita, vaan mahdollistaa myös joustavan ja kehittyvän käyttöympäristön, joka mukautuu tuleviin tarpeisiin ja muutoksiin.

Vaikka toiminnallisuuden ja käyttöliittymän suunnittelu oli vähäistä, täytyi kuitenkin tehdä päätöksiä ohjelmointikielen, alustan ja toteutustavan osalta. Ohjelmointikieleksi valikoitui JavaScript sen joustavuuden ja laajan käytettävyyden vuoksi. Erityisen tärkeä oli Node.js-ympäristön käyttö. Konkreettinen koodin kirjoittaminen suoritettiin Visual Studio Code -kehitysympäristössä, joka tarjoaa tarvittavat työkalut ja on kehittäjälle tuttu vuosien varrelta. Versionhallintaan päätettiin käyttää GitHubia, joka mahdollistaa muutosten helpon seurannan ja erilaisten kehityshaarojen käytön. Nämä päätökset tehtiin jo suunnitteluvaiheessa, jotta työn toteutus olisi mahdollisimman sujuvaa ilman turhia kokeiluja.

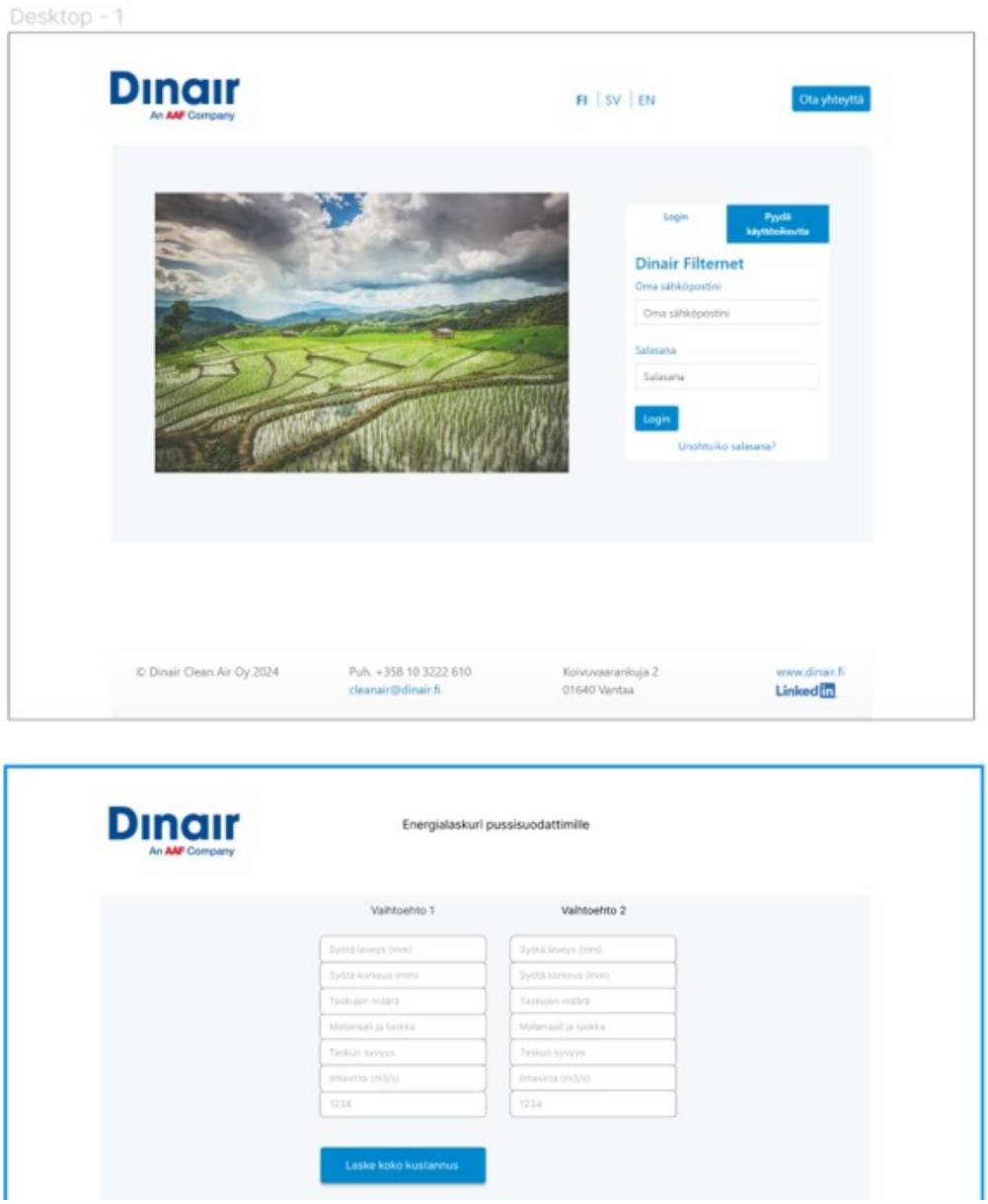
Jo suunnitteluvaiheessa ja myös työn alkuvaiheessa tuli ilmi muuttujien, funktioiden ja HTML-elementtien nimeämisen tärkeys. Kehitystyön selkeyttämiseksi on tärkeää olla yhtenäiset ja selkeät nimikäytännöt, jotta esimerkiksi arvot eivät sekoittuisi. Tämän lisäksi on hyvä ylläpitää koodin jatkuvaa kommentointia. Osuvat kommentit ja konsolikirjaukset auttavat

ongelmien selvittämisessä ja helpottavat koodin lukemista henkilöille, jotka eivät ole olleet mukana kehitystyössä.

Figma

Pilvipohjainen suunnittelutyökalu Figma soveltuu erinomaisesti kaikenlaisten sovellusten ulkoasun suunnitteluun. Figma-työkalun avulla pystytään luomaan interaktiivinen prototyyppi, jonka avulla voidaan mallintaa sovelluksen käyttökokemusta. Erilaisten linkityksien lisääminen näkymien välille mahdollistaa helpon siirtymisen näkymästä toiseen. Lisäksi työkalu sopii tiimityöskentelyyn, sillä useat käyttäjät voivat työskennellä samassa dokumentissa samanaikaisesti ja jakaa mielipiteitä sekä kommentteja suoraan Figma-alustalla. (What is Figma?).

Kuvassa 7 on esitetty ulkoasun suunnittelu uudelle energialaskurille. Kuvan 7 ensimmäinen näkymä kuvaa suunniteltua kirjautumissivua ja toinen näkymä on ensimmäinen suunniteltu näkymä laskurille. Kokonaisuudessaan jäi suunnittelutyö ulkoasun osalta kuitenkin vähäiseksi, sillä olemassa oleva Excel-tiedoston näkymä toimii jo varsin hyvänä pohjana uudelle ulkoasulle. Lisäksi Figma-työkalua ei käytetty sovelluksen käyttökokemuksen mallintamiseen, koska laskurin perusidea on hyvin yksinkertainen.



Kuva 7. Figman avulla tehty suunnittelu.

Web-sovelluksen tarkoituksena on siis vain syöttää laskuriin tarvittavat arvot, minkä jälkeen laskelmien tulokset esitetään käyttäjälle. Erilaisia vaihtoehtoja siirtymille tai sovelluksen perusrakenteelle ei tarvinnut testata. Figma-työkalun käytöstä on enemmän hyötyä, jos suunnitellaan ulkoasu ja siirtymät ilman minkäänlaista pohjaa tai muutokset olisivat todella isoja.

5 Toteutus

Web-sovelluksen tekeminen aloitettiin rakentamalla sovellukselle perusrakenne. Ensimmäinen askel oli varmistaa Node.js:n asennus ja versio. Asennetun version tarkistamiseksi käytettiin komentorivillä komentoa ”node -v”. Käyttämällä npm-työkalua päivitettiin Node.js uusimpaan versioon. Tämän jälkeen täytyi luoda projekti käyttämällä komentoa ”npm init” ja lisätä projektiin Express.js, joka luo sovellukselle peruspohjan, joka sisältää tarvittavat rakenteet ja tiedostot. Seuraavaksi luotiin app.js-niminen tiedosto, jonka alkuperäinen sisältö on esitetty kuvassa 10. Tälle peruspohjalle pystyy helposti rakentamaan ulkoasua ja toimintoja.

```
const express = require('express');

const app = express();

app.get('/', (req, res) => {

    res.send('Hello World!');

});

app.listen(3000, () => {

    console.log('Server listening on port 3000');

});
```

Esimerkkikoodi 2. App.js-tiedoston alkuperäinen sisältö (Khan 2023).

5.1 Tietokanta ja MariaDB

Tietokannan luomiseen käytettiin MySQL:ään pohjautuvaa relaatiotietokantajärjestelmää MariaDB:tä. Tietokantapyyntöihin käytetään SQL-kieltä (Structured Query Language), joka on todella yleisesti käytössä muissakin järjestelmissä. MariaDB valikoitui tähän projektiin, koska se on ennestään tuttu ja soveltuu kaikenkokoisille projekteille.

MariaDB:n asentamisen jälkeen luotiin uusi tietokanta nimeltä Energialaskuri, johon luotiin kaksi taulua. Taulut sisältävät olemassa olevan energialaskurin datan pussisuodatin-malleista. Seuraavaksi oli tärkeää luoda yhteys web-sovelluksen ja tietokannan välille. Yhteyden luominen tapahtuu connectToDatabase-nimisellä funktiolla, jossa käytetään Node.js:n moduulin pakettia mysql12.

Jotta laskelmia pystytään suorittamaan, sovelluksen täytyy hakea tietoja tietokannasta. Laskutoimitukset eivät ainoastaan toteudu käyttäjän syöttämän tiedon perusteella, vaan käyttäjän syöttämälle mallille haetaan vastaavat arvot tietokannasta. Tämän lisäksi sovellus hakee tietokannasta esitettävää tietoa.

Esimerkkikoodissa 3 näkyy getProductDesc-niminen funktio, jolla muun muassa luodaan yhteys tietokantaan. Select-tietokantakyselyllä haetaan Products-taulusta kaikki rivit product_type-sarakkeesta ja esitetään jokainen ainutlaatuinen tietue listassa kerran. Kyseistä funktiota käytetään energialaskurin laskuri-näkymässä, jotta käyttäjän on helppo valita suodatinmateriaali ja -luokka. Suurin osa tietokantakyselyistä, joita tässä toteutuksessa suoritetaan, on tiedon hakemiseen tarkoitettuja. Muita samantyyppisiä funktioita, jotka sisältävät SQL-kyselyitä, ovat muun muassa getProductInfo ja getMaterialThickness.

```
async function getProductDesc() {
  try {
    const connection = await connectToDatabase();
    const [rows] = await connection.execute(
      'SELECT product_type FROM Products',
    );
    const data = rows;
    connection.end();
    const uniqueProductTypes = [
      ...new Set(data.map((item) => item.product_type)),
    ];
    const uniqueOptions = uniquePro-
ductTypes.map((productType) => {
      return { product_type: productType };
    });
    return uniqueOptions;
  } catch (error) {
    console.error('Error querying database:', error);
    throw error;
  }
}
```

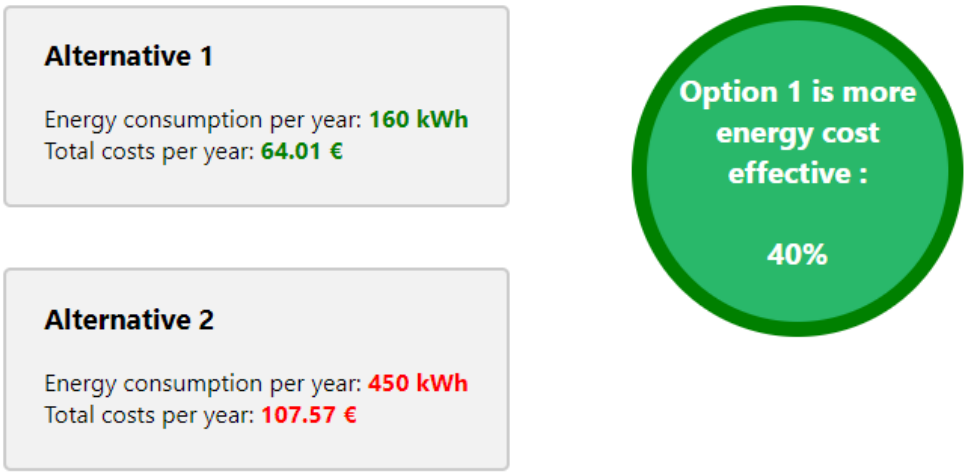
Esimerkkikoodi 3. getProductDesc funktio.

Tietokannassa tiedot eivät ole niin sanotusti kovakoodattuja eli ne ovat helposti muokattavissa. Lisäksi tavallisella käyttäjällä ei ole pääsyä koko tietokantaan. Tiedon säilyttäminen tietokannassa edistää sovelluksen tietoturvaa, jos tiedonsäilytystapaa vertaa edelliseen toteutukseen. Tietokanta-arvojen muokkaaminen, poistaminen ja lisääminen kuuluvat tämän toteutuksen tulevaisuusnäkymään. Tällä hetkellä tietokantamuutokset tapahtuvat suoraan tietokannasta komentoriviä käyttäen.

5.2 Laskurin toiminnallisuus

Yksikertaistetusti energialaskuri kerää käyttäjän antamat tiedot form-elementissä olevilla input-kentillä. Form-elementin (lomakkeen) toiminnoksi (action) on asetettu ”/calculate”. Kun lomakeessa olevaa nappia napsauttamalla, suoritetaan ”/calculate”-niminen POST-pyyntö. Kyseisessä pyynnössä suoritetaan laskutoiminnot useilla funktioilla, jotka on tarkemmin määritelty erillisessä tiedostossa nimeltä functions.js. Tiedostossa olevia funktioita ovat esimerkiksi calculateTotalPrice, calculateDifference ja calculateYearlyConsumption. Oikeassa muodossa olevat tulokset lähetetään res.render-toiminnolla takaisin calculator-näkymään suoraan käyttäjälle.

Ensiksi käynnistyy getProduct-niminen funktio, joka hakee käyttäjän syöttämien arvojen (suodatinmateriaali, pussin pituus, taskumäärä) perusteella funktion käyttöön yhdistelmään kuuluvat painehäviö-arvot. Nämä kolme parametriä sopivat selvittämään yksilöllisen tuloksen tietokannasta. Tämän jälkeen suoritetaan useita laskelmia ja tarkastellaan syötettyjä arvoja . Esimerkiksi käyttäjän syöttämää taskumäärää ei välttämättä aina käytetä suoraan laskelmissa, sillä todelliseen taskumäärä-arvoon vaikuttaa myös laskelmaan käytetty suodatinmateriaali. Lasketaan siis laskennallinen pussin leveys, johon on määrätty tietty vastaava vakiosuodattimen pussimäärä. Laskelmien viimeiset, lopulliset arvot esitetään käyttäjälle, jolloin hän näkee tulokset ja vertailuarvot. Kuvassa 8 näkyy uuden toteutuksen esimerkki tulossivu, jonka myös käyttäjä voi ladata halutessaan PDF-tiedostona.



Kuva 8. Uuden toteutuksen tulossivu.

Energialaskurin toiminnallisuuteen kuuluu vaatimus, että jokainen kenttä on täytettävä ennen laskennan aloittamista. Mikäli käyttäjä jättää jonkin kentän tyhjäksi, sovellus ilmoittaa siitä pop-up-ilmoituksella ja estää laskennan käynnistymisen. Vaatimuksella varmistetaan, että laskelmat perustuvat täydellisiin tietoihin ja estävät virheellisten tulosten esittämisen. Tämän lisäksi kentille on asetettu myös rajoituksia, jotka estävät laskennan aloituksen ja antavat tietoa myös käyttäjälle. Esimerkiksi kenttiin leveys ja korkeus pystyy vain kirjoittamaan numeroita. Jos käyttäjä kuitenkin syöttää kenttään esimerkiksi sanan tai erikoismerkin, näytölle tulee ilmoitus, jossa ilmoitetaan väärällä tiedolla oleva kenttä ja siihen korjausehdotus. Käyttötuntimäärä-kenttään pätee myös sääntö, joka estää syöttämästä tuntimäärää alle nollan tai yli 8 765:n. Oletusarvo käyttötuntimäärään on 6 000 tuntia, joka on myös standardilaskelmiin käytetty Euroventin arvo. Määrää voi siis muutella, mutta oletukseksi on asetettu yleisin laskentaan käytetty arvo, mikä edistää helppokäyttöisyyttä.

5.3 Web-sovelluksen käyttöliittymä

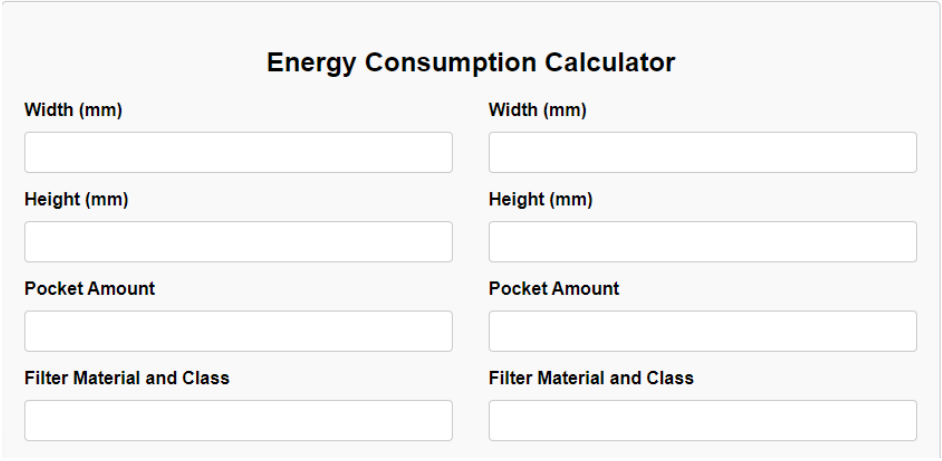
Käyttöliittymän rakentaminen aloitettiin luomalla index.ejs-tiedosto, johon Express.js mahdollistaa helpon hahmontamisen. Ensimmäiseksi luotiin HTML form -elementillä laskurille perusrakenne, joka sisältää erilaisia kenttiä ja alasvetovalikkoja. CSS-tyylittely jäi varsin minimaaliseksi, kun kehityksen pääkohteena olivat laskurin toiminnallisuudet ja sitä mahdollistavat funktiot.

Esimerkkikoodissa 4 on esitetty Express.js:lle tyypillinen hahmontaminen, joka on toteutettu EJS-näkymämoottorin avulla. Express.js hahmontaa kuvassa näkyvän HTML-koodin dynaamisesti käyttäen EJS-muuttujia ja -säätimiä (”<% %>”). Kun Express.js saa pyynnön, se lähettää pyynnön HTML-koodin vastauksena ja suorittaa EJS-syntaksin, jolloin ”<%= item.product_type %>”-kohdassa näytetään dynaamisesti tietokannasta haetut ”item.product_type”-arvot pudotusvalikkoon. Samantyylisesti esitetään myös laskelmien tulokset. Näkymämoottoreiden, kuten EJS:n, avulla voidaan helposti erottaa sivujen ulkoasu ja dynaaminen sisältö toisistaan.

```
<h2>Alternative 1</h2>

<label for="dbValue1">Filter material and class: </label>

<select class="js-example-basic-single" style="width:
100%;" name="dbValue1">>

   <option value="">Choose: </option>

   <% uniqueOptions.forEach(item => { %>

<option value="<%= item.product_type %>"><%= item.prod-
uct_type %></option> <% }); %></select>
```

Esimerkkikoodi 4. EJS-näkymämoottorin avulla toteutettu hahmontaminen.

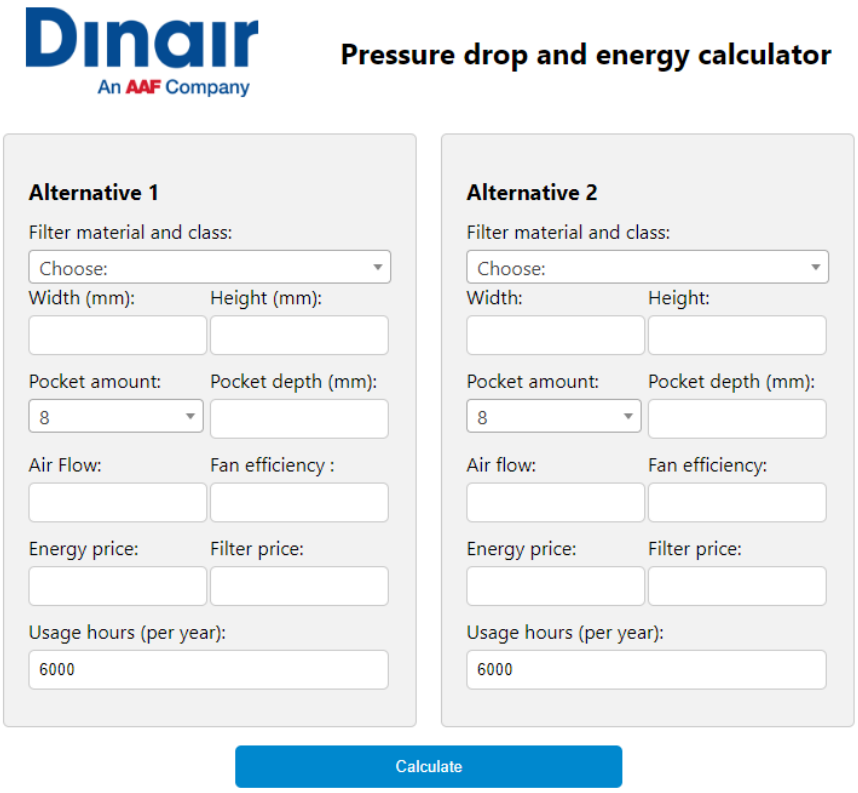
Kuvassa 9 näkyy laskurin ulkoasu kehitystyön alkuvaiheessa ilman ulkoasullisia viimeistelyjä.



Kuva 9. Uuden toteutuksen alkuvaiheen laskurinäkymä.

Kuvassa 9 kenttien muotoilu, näkymän värimaailma ja fontit ovat tässä vaiheessa todella yksinkertaisia eivätkä ne sovi yhteen yrityksen verkkosivujen ulkoasun kanssa. Input-kentät ovat allekkain vaaleanharmaalla taustalla.

Kuvassa 10 energianlaskurin ulkoasu on enemmän yrityksen brändin mukainen. Ulkoasu sisältää yrityksen logon sekä sopivat fontti- ja väriratkaisut. Yleinen ulkoasutyyli on yksinkertainen ja minimalistinen, mutta yhteensopiva ja tyylikäs. Värimaailmana on käytetty harmaata, mustaa ja sinistä.



Kuva 10. Uuden toteutuksen loppuvaiheen laskurinäkymä.

Lisäksi uuden toteutuksen tulossivulla on käytetty värikoodeina punaista ja vihreää, jotta energiakustannustehokkaampi vaihtoehto erottuisi helpommin.

6 Yhteenveto

Tämän insinöörityön tarkoituksena oli kehittää toimeksiantajalle, Dinair Clean Air Oylle, toimiva web-sovellus, jonka inspiraationa ja pohjana toimii käytössä oleva Excel-pohjainen laskuri. Keskeisenä painopisteenä oli laskurin käyttäjäystävällisyyden parantaminen ja pohjan rakentaminen uusille kehitysmahdollisuuksille. Vaikka nyt toteutettu web-sovellus toimii perusominaisuuksiltaan hyvin, sen laajempi käyttöönotto myyjien ja asiakkaiden keskuudessa vaatii vielä lisää kehitystyötä. Sovelluksen integraatio osaksi olemassa olevaa asiakasportaalia auttaa ratkaisemaan nykyiset tietoturvahaasteet ja mahdollistaa sovelluksen laajemman käytön verrattuna nykytilanteeseen. Tulevaisuusnäkymään kuuluu myös ominaisuuksien, kuten

ilmanvaihtokoneen ja kiinteistökohtaisen laskurin lisääminen ja ulkoasun viimeistely.

Insinöörityössä onnistuttiin tasapainoisessa ajankäytössä kirjoitustyön ja sovelluskehitystyön välillä, kattavan teoriapohjan rakentamisessa ja sovelluksen päätoimintojen toteuttamisessa. Lisäksi toimeksiantajan eli nykyisen työpaikan kanssa kommunikointi oli helppoa ja rakentavaa. Tämä työ opetti siis paljon yrityksen sisäisestä toiminnasta, kommunikaatiosta ja mahdollisuuksista.

Sovelluskehitystyön haasteiksi osoittautuvat ajan puute, laaja kokonaisuus ja suuret tavoitteet sekä tekniset haasteet. Kuitenkin työn jokaisen vaiheen suorittaminen yksin eikä esimerkiksi ryhmätyönä oli erittäin opettavainen ja hyödyllinen kokemus tulevaisuuden projekteja ajatellen. Toimeksiantajan näkökulmasta tämä insinöörityön puitteissa tehty toteutus oli riittävä, kun tarkastelee työhön käytettyä aikaa.

Sovelluskehitystyön prosessi antoi arvokasta kokemusta monipuolisista teknisistä haasteista ja niiden ratkaisemisesta. Työskentely Dinair Clean Air Oyn kanssa antoi laajaa ymmärrystä yrityksen tarpeista ja toimintaympäristöstä. Oli rohkaisevaa työskennellä projektissa, jolla on selkeä käyttötarkoitus ja merkitys jokapäiväiseen toimintaan.

Lähteet

About Node.js. Verkkoaineisto. <https://nodejs.org/en/about>. Luettu 31.3.2024.

Agarwal, Sudhanshu. Best Practices for Username and Password Authentication. Verkkoaineisto. Loginradius. <https://www.loginradius.com/blog/identity/best-practices-username-password-authentication/>. Luettu 9.4.2024.

An introduction to the npm package manager. Verkkoaineisto. <https://nodejs.org/en/learn/getting-started/an-introduction-to-the-npm-package-manager>. Luettu 31.3.2024.

Cairns, Cade & Somerfield Daniel. 2017. The Basics of Web Application Security. Verkkoaineisto. <https://martinfowler.com/articles/web-security-basics.html>. 5.1.2017. Luettu 7.4.2024.

Craig, Julie; Pollard, Carol & Strum, Rick. 2017. Application Performance Management (APM) in the Digital Enterprise. E-kirja. Elsevier Inc. 24.2.2017.

Dabbs, Mark. 2019. The Fundamentals of Web Application Architecture. Reinvently blogi. <https://reinvently.com/blog/fundamentals-web-application-architecture/> 29.6.2019. Luettu 2.4.2024.

Halili, Festim & Ramadani, Erenis. 2018. Web Services: A Comparison of Soap and Rest Services. Verkkoaineisto. Canadian Center of Science and Education <https://www.researchgate.net/publication/323456206_Web_Services_A_Comparison_of_Soap_and_Rest_Services>. 28.2.2018. Luettu 31.3.2024.

Identification and Authentication Failures. 2021. Verkkoaineisto. OWASP Top 10:2021. <https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/>. 2021. Luettu 7.4.2024.

Khan, Shahzaib. 2023. Building Web Applications with Express.js: A Comprehensive Guide. Verkkoaineisto. Medium. <https://medium.com/@skhans/building-web-applications-with-express-js-a-comprehensive-guide-113a77be1b11>. 1.3.2023. Luettu 27.4.2024.

Laine, Harri. 1999. Tietokantojen perusteet, osa 1. Opetusmoniste. Helsingin yliopisto, tietojenkäsittelytieteen laitos. Luettu 2.4.2024.

Murphy, Natasha. 2024. What is Database Security? Common Threats & Best Practices. Verkkoaineisto. Lepide. <https://www.lepide.com/blog/what-is-database-security/>. 29.2.2024. Luettu 2.4.2024

Opi MySQL/MariaDB aloittelijoille - Osa 1. Verkkoaineisto. <https://fi.linux-console.net/?p=1619#gsc.tab=0>. Luettu 2.4.2024.

Password Hashing and Salting Explained. 2023. Verkkoaineisto. Authgear. <https://www.authgear.com/post/password-hashing-salting>. 14.7.2024. Luettu 7.4.2024.

Tomar, Anuj. 2023. 10 Best Web Development Frameworks to Use in 2024. Verkkoaineisto. Medium. <https://medium.com/@anuj.t_8752/10-best-web-development-frameworks-to-use-in-2024-updated-ff988e8f85cb>. 4.12.2024. Luettu 3.4.2024.

Volle, Adam. 2022. Web Application. Verkkoaineisto. Encyclopedia Britannica. <https://www.britannica.com/technology/spreadsheet>. 6.10.2022. Luettu 31.3.2024.

What is Figma? Verkkoaineisto. <https://help.figma.com/hc/en-us/articles/14563969806359-What-is-Figma>. Luettu 27.4.2024.

6 types of security vulnerabilities in e-commerce and how to solve them. 2020. Verkkoaineisto. N-iX. <https://www.n-ix.com/6-types-security-vulnerabilitiesecommerce-solve-them/>. 23.8.2023. Luettu 9.4.2024.