VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Kasper Stenfors

KALENTERI- JA MUISTIINPANO-SOVELLUS

Tekniikka
2025

VAASAN AMMATTIKORKEAKOULU
Tietotekniikka

TIIVISTELMÄ

Tekijä	Kasper Stenfors
Opinnäytetyön nimi	Kalenteri- ja muistiinpanosovellus
Vuosi	2025
Kieli	suomi
Sivumäärä	52
Ohjaaja	Mikael Jakas

Tämän opinnäytetyön päätavoitteena oli kehittää web-pohjainen kalenteri- ja muistiinpanosovellus, joka tarjoaa käyttäjälle helppokäyttöisen työkalun ajanhallinnan ylläpitämiseen ja henkilökohtaisten muistiinpanojen tallentamiseen. Sovellus on web-pohjainen ja se toimii sekä tietokoneella että mobiililaitteilla. Tavoitteena oli myös toteuttaa kirjautuminen, jotta jokaisella käyttäjällä olisi oma sisältönsä.

Sovellus tehtiin Node.js-ympäristössä käyttäen Express-kehystä. Käyttäjätiedot ja tapahtumat tallennetaan MariaDB-tietokantaan. Salasanat suojataan bcrypt-kirjastolla ja kirjautuminen toteutetaan JSON Web Token (JWT:n) avulla. Käyttöliittymä on tehty HTML:llä, CSS:llä ja JavaScriptillä. Työssä käytettiin apuna myös ympäristömuuttujia (.env) tietoturvan parantamiseksi. Tekstieditorina toimi Visual Studio Code.

Sovelluksen toimivuutta testattiin manuaalisesti. Käyttäjä voi kirjautua, luoda tapahtumia ja kirjoittaa muistiinpanoja omalle tililleen. Tavoitteet saavutettiin hyvin. Sovelluksen rakenne on suunniteltu siten, että siihen on helppo lisätä uusia ominaisuuksia myöhemmin. Tämä tekee sovelluksesta hyvin skaalautuvan ja ylläpidettävän.

Avainsanat ohjelmointi, kalenterisovellus, muistiinpanot, web-kehitys

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Tietotekniikka

ABSTRACT

Author	Kasper Stenfors
Title	Calendar and Notes Application
Year	2025
Language	Finnish
Pages	52
Name of Supervisor	Mikael Jakas

The main goal of this thesis was to develop a web-based calendar and notes application that provides users with an easy-to-use tool for managing their time and storing personal notes. The application is browser-based and works on both desktop and mobile devices. One of the key objectives was to implement a login system, allowing each user to access their own content.

The application was developed in a Node.js environment using the Express framework. User data and events are stored in a MariaDB relational database. Passwords are protected using the bcrypt library, and user authentication is handled with JSON Web Tokens (JWT). The user interface was built with HTML, CSS, and JavaScript. Environment variables (.env) were used to enhance security, and the development was carried out in Visual Studio Code.

The functionality of the application was tested manually. Users can log in, create events, and write personal notes in their own accounts. The project goals were achieved. The structure of the application was designed to be easily extendable, making it scalable and maintainable for future development.

Keywords	programming, application, software development, planning and design

SISÄLLYS

KUVAT

LYHENTEET

API	Application programming Interface
CSS	Cascading Style Sheets
DB	Database
EJS	Embedded Javascript Templates
HTML	Hypertext Markup Language
ESM	ECMAScript Modules
HTTP	Hypertext Transfer Protocol
JWT	JSON Web Token
npm	Node Package Manager
REST	Representational State Transfer
SQL	Structured Query Language
AJAX	Asynchronous JavaScript and XML
XSS	Cross Site Scripting

1 JOHDANTO

Kalenteri- ja muistiinpanosovellusten tarkoituksena on helpottaa arjen organisointia ja ajanhallintaa. Ne tarjoavat käyttäjille mahdollisuuden kirjata ylös tulevia tapahtumia, aikatauluja ja tärkeitä muistettavia asioita. Erityisesti mobiililaitteilla toimivat sovellukset ovat lisänneet ajanhallinnan saavutettavuutta, sillä niitä voidaan käyttää missä ja milloin tahansa.

Tämän opinnäytetyön tavoitteena oli toteuttaa selainpohjainen kalenteri- ja muistiinpanosovellus, jota voi käyttää sekä tietokoneella että mobiililaitteilla. Sovellus mahdollistaa käyttäjän rekisteröitymisen ja kirjautumisen omaan tiliinsä, tapahtumien lisäämisen sekä henkilökohtaisten muistiinpanojen tallentamisen tietokantaan. Työn painotus oli käytännön toteutuksessa: käyttäjäkirjautumisen turvallisuudessa, tietojen tallentamisessa sekä selkeän ja responsiivisen käyttöliittymän rakentamisessa. Sovellus tehtiin itsenäisesti ilman ulkopuolista tilaajaa, ja sen rakenteessa huomioitiin myös mahdollisuudet laajentamiseen tulevaisuudessa.

Tämän opinnäytetyön aikana olen hyödyntänyt ChatGPT:tä tukena erityisesti ideoinnissa, tekstin kieliasun parantamisessa sekä ohjelmointiapuna. Tekoälyä käytettiin tukemaan työn etenemistä ja kehittämään ilmaisua, mutta lopullinen sisällön tuottaminen, muokkaaminen ja tarkastaminen on tehty omatoimisesti.

Jos tekoäly on tuottanut tekstiin uusia ideoita, olen aina tarkistanut ne alkuperäisistä lähteistä ja viitannut niihin asianmukaisesti. Ohjelmoinnissa tekoäly auttoi minua bugien etsimisessä, koodin selittämisessä ja ymmärtämisessä sekä parhaiden käytäntöjen löytämisessä esimerkiksi tietoturvan ja responsiivisuuden toteuttamiseksi. Kaikki saadut neuvot ja ehdotukset on kuitenkin tarkistettu itsenäisesti luotettavista lähteistä,

jotta ne pitävät paikkaansa. Olen käyttänyt sitä vastuullisesti myös tietosuojan kannalta ja huolehtinut siitä, että työn sisältö vastaa omaa ajattelua.

Tekoälyn käyttö tässä projektissa on auttanut tekemään prosessista sujuvampaa ja tarjonnut tukea, jonka avulla olen pystynyt kehittämään sekä teknisiä että kirjallisia taitojani opinnäytetyön aikana.

2 TEKNOLOGIAT

Tässä luvussa käydään läpi sovelluksessa käytetyt teknologiat ja työkalut, jotka yhdessä muodostavat modernin web-sovelluksen arkkitehtuurin.

2.1 Node.js, Express ja REST-rajapinta

Sovelluksen palvelin toteutettiin käyttäen Node.js:ää ja Express-kehystä (Express framework). Node.js on avoimen lähdekoodin ajoympäristö, joka mahdollistaa JavaScriptin suorittamisen palvelimella eli selainympäristön ulkopuolella. Se perustuu Chromen V8 JavaScript -moottoriin, ja sitä käytetään yleisesti backend-kehitykseen (Node.js, n.d.). Tämä tekee sovelluskehityksestä tehokkaampaa, kun tarvitsee käyttää vain yhtä ohjelmointikieltä.

Express on puolestaan Node.js:lle rakennettu kevyt ja joustava web-kehys, joka helpottaa palvelinpuolen sovellusten kehittämistä. Se tarjoaa yksinkertaisen tavan käsitellä HTTP-pyyntöjä ja määritellä reittejä, minkä ansiosta sovelluksen rakenne pysyy selkeänä ja ylläpidettävänä. Expressiä käytetään laajalti modernien verkkosovellusten ja REST-rajapintojen rakentamisessa (Express, n.d.-b). Expressin avulla sovellukseen pystyi helposti määritellä eri reitit, kuten tapahtumien lisääminen, muokkaaminen ja poistaminen.

Sovelluksen reititys noudattaa REST-arkkitehtuurin periaatteita. REST (Representational State Transfer) on yksinkertainen ja selkeä tapa rakentaa verkkopalveluita. Sen ideana on, että sovelluksen eri resurssit, kuten kalenteritapahtumat ja muistiinpanot voidaan hakea ja muokata standardoitujen HTTP-pyyntöjen avulla (Mozilla Developer Network, n.d.-c). Express ja REST toimivat hyvin yhdessä, sillä Express tarjoaa yksinkertaiset työkalut juuri tällaisen REST-tyylisen reitityksen toteuttamiseen.

2.2 Tietokanta

Sovelluksessa käytettiin tietojen tallentamiseen MariaDB-relaatiotietokantaa. MariaDB on avoimen lähdekoodin tietokantajärjestelmä, joka perustuu MySQL:n lähdekoodiin (MariaDB Foundation, n.d.-a). Se on yhteensopiva monien ohjelmointikielten ja työkalujen kanssa, ja sitä käytetään laajasti eri kokoisissa ohjelmistoprojekteissa.

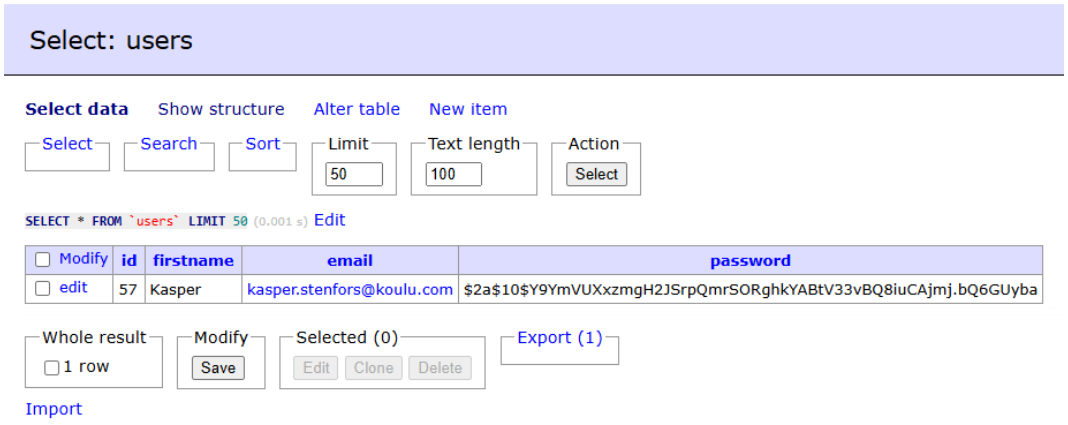
Tietokantaan tallennettiin käyttäjien tiedot, tapahtumat sekä muistiinpanot. Jokaiselle kirjautuneelle käyttäjälle tallennettiin oma data tietokantaan käyttäjän tunnisteen perusteella. Näin eri käyttäjien tiedot pysyvät erillään ja sovellus voi näyttää vain kirjautuneen käyttäjän omat tapahtumat ja muistiinpanot.

Node.js:ssä tietokantayhteys toteutettiin mariadb-nimisen npm-kirjaston avulla (MariaDB Foundation, n.d.-b). Tämä kirjasto mahdollistaa helpon tavan muodostaa yhteys ja suorittaa SQL-kyselyitä etäpalvelimella toimivaan MariaDB-tietokantaan. Tietokantarakenteen suunnittelussa käytettiin relaatiomallia, jossa tiedot tallennettiin tauluihin. Jokaisella käyttäjällä on yksilöllinen ID-tunniste, jota käytetään viitteellisenä avaimena tapahtuma- ja muistiinpanotietueissa. Tämä rakenne tekee tietojen hakemisesta ja käsittelystä tehokasta ja loogista. Yhteys määriteltiin erillisessä db.js-tiedostossa, jossa käytettiin ympäristömuuttujia (.env) tietoturvan varmistamiseksi (Motdotla, n.d.). Ympäristömuuttujien käyttö suojaa tärkeitä tietoja päätymästä näkyviin versionhallintaan, kuten GitHubiin.

2.3 Tietoturva

Koska sovelluksessa käyttäjät kirjautuvat sisään ja näkevät henkilökohtaisia tietojaan, tietoturva on tärkeässä roolissa. Tavoitteena oli varmistaa, että arkaluonteinen tieto, kuten salasanat, käsitellään turvallisesti eikä niitä tallenneta selväkielisinä tietokantaan.

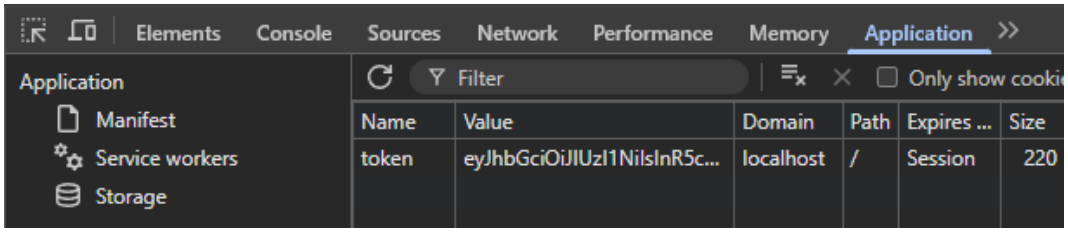
Salasanojen käsittelyssä käytettiin bcryptjs-kirjastoa, jonka avulla käyttäjän salasana muunnetaan hash-muotoon ennen tallentamista (bcrypt.js, n.d.). Kuvassa 1 näkyy salasana hashattuna tietokannassa.



Kuva 1. Esimerkki hashatusta salasanasta MariaDB-tietokannassa.

Hashaus on yksisuuntainen prosessi, eli alkuperäistä salasanaa ei voi palauttaa hashista takaisin. Tämä estää salasanojen vuotamisen silloinkin, jos tietokantaan päästäisiin käsiksi. Kun salasana muunnetaan hash-muotoon, siihen lisätään satunnainen merkkijono eli suola (salt). Tämä suojaa tehokkaasti sanakirjahyökkäyksiä ja rainbow table -hyökkäyksiä vastaan. Tässä projektissa käytettiin kymmenen kierroksen suolausta (salt = 10), mikä tarkoittaa, että bcrypt suorittaa suolauksen ja hashausprosessin useita kertoja ennen lopullisen hashin muodostamista (OWASP, n.d.-b).

Kun käyttäjä kirjautuu sovellukseen onnistuneesti, hänelle luodaan jsonwebtoken-kirjastolla JWT-tunnus (JSON Web Token, Auth0, n.d.). Tunnus sisältää käyttäjän tunnistetiedot ja se allekirjoitetaan salaisella avaimella. Token toimitetaan sitten käyttäjälle ja tallennetaan selaimen evästeeseen. Kuvassa 2 nähdään JWT-tunniste tallennettuna selaimen evästeeseen Google Chromen kehittäjätyökalussa.



Kuva 2. JWT-tunniste tallennettuna selaimen evästeeseen. (Kuva Google Chromen kehittäjätyökalusta).

JWT-tunniste mahdollistaa sen, että käyttäjä voidaan tunnistaa jatkossa jokaisella pyynnöllä ilman uutta kirjautumista. Sovellus käyttää cookie-parser-kirjastoa, jonka avulla Express osaa lukea selaimelta saapuvat evästeet ja tarkistaa JWT-tunnuksen oikeellisuuden (Express, n.d.-a). Näin voidaan varmistaa, että pyyntö tulee oikealta käyttäjältä. Kuten tietokantayhteyden tunnukset, myös JWT:n salainen avain säilytetään ympäristömuuttujana .env-tiedostossa.

2.4 Käyttöliittymä

Sovelluksen käyttöliittymä toteutettiin HTML-, CSS- ja JavaScriptin avulla. Käyttöliittymän tavoitteena oli tarjota käyttäjälle selkeä ja helppokäyttöinen tapa tarkastella kalenterinäkymää, lisätä tapahtumia ja kirjoittaa muistiinpanoja. Koska käyttöliittymä toimii tietokoneella kuin

myös mobiililaitteilla, sen suunnittelussa kiinnitettiin huomiota responsiivisuuteen.

HTML:n avulla rakennettiin sivujen rakenne ja elementit, kuten lomakkeet, kalenteripohja ja muistiinpanokentät. CSS:ää käytettiin sivujen ulkoasun muotoiluun, kuten värityksiin, asetteluihin ja fonttityyleihin. Responsiivisuus toteutettiin media queries-tekniikalla sekä käyttämällä skaalautuvia yksiköitä, kuten rem (Mozilla Developer Network, n.d.-b). Näiden avulla sisältö saatiin mahtumaan näytölle ja pysymään miellyttävänä erikokoisilla laitteilla.

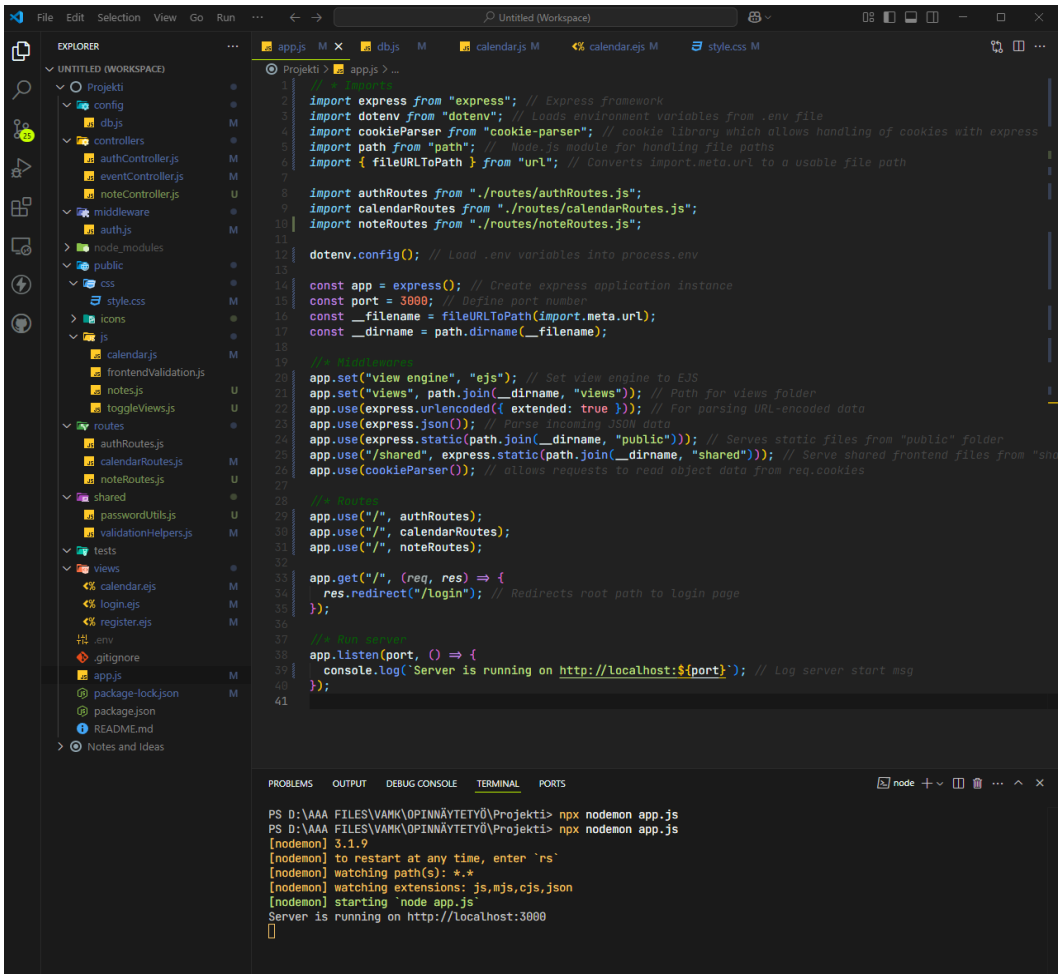
JavaScriptillä rakennettiin käyttöliittymän vuorovaikutteiset toiminnot, kuten tapahtumien ja muistiinpanojen lisääminen, muokkaaminen ja poistaminen, kalenterin päivittäminen sekä näkymien vaihtaminen ilman sivun uudelleenlatausta. Lisäksi JavaScriptillä hallitaan lomakkeiden validointia, jotta mahdolliset virheet voidaan havaita jo ennen tietojen lähettämistä palvelimelle.

Sovelluksessa sivut renderöidään palvelinpuolella EJS:n (Embedded JavaScript) avulla. EJS on templating-moottori, joka mahdollistaa muuttujien ja dynaamisen datan lisäämisen HTML-sivuille (EJS, n.d.). Tässä projektissa EJS:ää käytettiin kuitenkin lähinnä kirjautuneen käyttäjän etunimen sekä virheilmoitusten näyttämiseen käyttöliittymässä.

2.5 Kehitysympäristö

Sovellus kehitettiin Visual Studio Code -editorilla, joka on kevyt ja laajasti käytetty ohjelmointiympäristö erityisesti web-kehityksessä. Editorin valintaan vaikutti sen aikaisempi tuttuus, helppokäyttöisyys, laajat laajennusmahdollisuudet sekä sisäänrakennetut ominaisuudet, kuten Git-versionhallinta (Visual Studio Code, n.d.). Projektissa käytettyjä laajennuksia on esimerkiksi Prettier, joka huolehti automaattisesta koodin muotoilusta aina tiedoston tallennuksen yhteydessä (Prettier, n.d.) sekä

Better Comments, joka mahdollisti värikoodatut kommentit, joilla pystyy helposti erottelemaan erilaiset tehtävät, kuten tärkeät, bugit ja varoitukset toisistaan (Aaron Bond, n.d.). Kuvassa 3 nähdään kuvankaappaus Visual Studio Code -editorista.

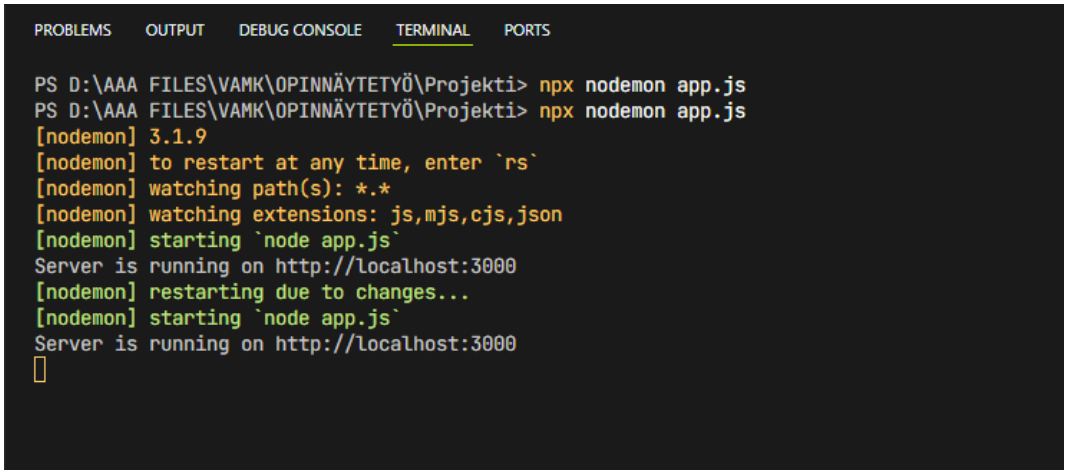


Kuva 3. Kuvankaappaus Visual Studio Code -editorista.

Projektin versiohallinnassa käytettiin Git-työkalua sekä GitHub-palvelua. Git mahdollistaa muutosten seuraamisen ja hallitsemisen koodissa, ja

GitHubin avulla projekti voitiin tallentaa pilveen sekä työstää sitä esimerkiksi kannettavalta tietokoneelta. Tämä mahdollisti myös turvallisen varmuuskopioinnin projektin edetessä (GitHub, n.d.).

Sovellusta ajettiin paikallisesti Node.js-ympäristössä, ja kehitystyötä nopeutettiin nodemon-kirjaston avulla. Kuvasta 4 nähdään, kuinka nodemon tarkkailee projektin tiedostoja ja käynnistää palvelimen automaattisesti uudelleen tiedostojen muuttuessa (Remy Sharp, n.d.).



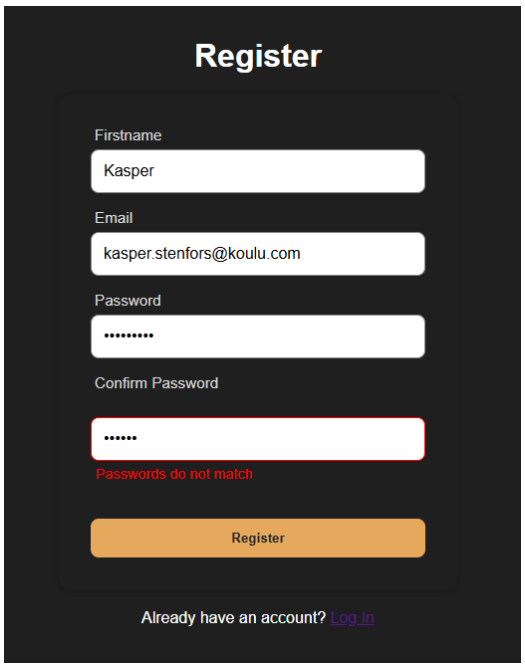
Kuva 4. Terminaalinäkymä, jossa nodemon uudelleenkäynnistää ohjelman tiedostojen muuttuessa.

3 SOVELLUKSEN KUVAUS

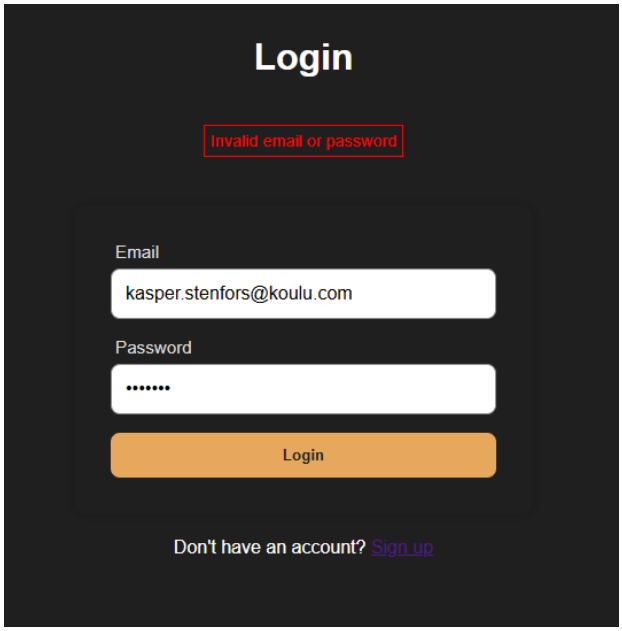
Tässä luvussa esitellään sovelluksen toiminnot ja näkymät käyttäjän näkökulmasta. Luvussa käydään läpi rekisteröityminen, kirjautuminen sekä kalenterin ja muistiinpanojen käyttö. Lisäksi kuvataan sovelluksen käytettävyyttä mobiililaitteilla.

3.1 Rekisteröinti ja kirjautuminen

Kun käyttäjä saapuu ensimmäistä kertaa sovelluksen aloitussivulle, hänen tulee rekisteröityä. Rekisteröityessä käyttäjältä pyydetään nimi, sähköposti ja salasana. Virhetilanteissa, kuten väärän salasanan syötön jälkeen, käyttäjälle annetaan ilmoitus siitä, mikä meni pieleen ja mitä tulee korjata. Onnistuneen rekisteröitymisen jälkeen käyttäjä ohjataan kirjautumissivulle, ja kirjautumisen jälkeen suoraan kalenterinäkymään. Kuvissa 5 ja 6 on esimerkkejä virhetilanteista.



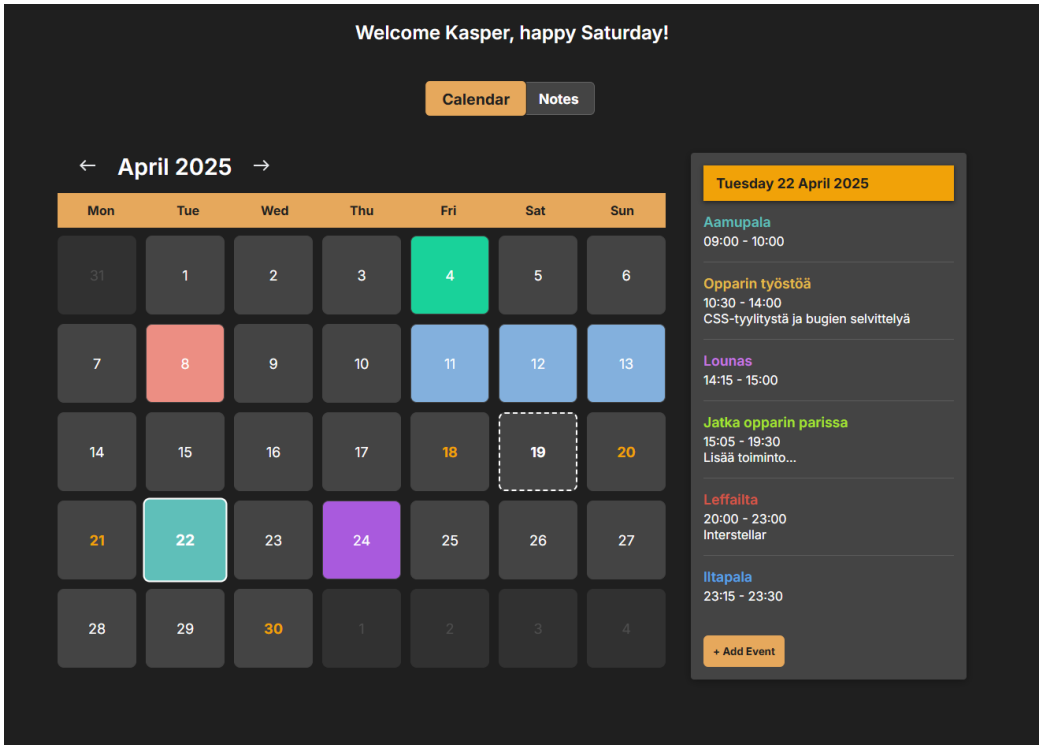
Kuva 5. Rekisteröintisivu ja virhe salasanan varmistuksessa.



Kuva 6. Kirjautumissivu ja väärillä tunnuksilla kirjautuminen.

3.2 Kalenterinäkymä

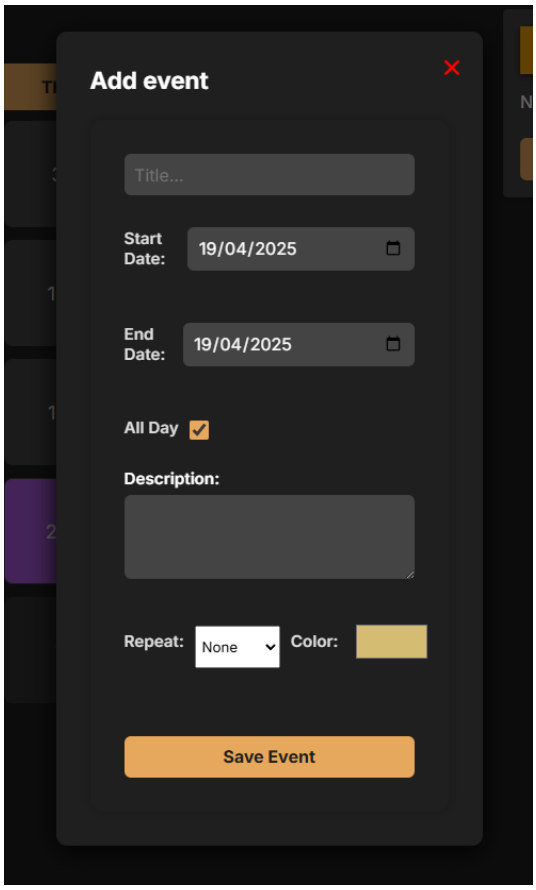
Kalenterinäkymässä käyttäjä näkee kuluvan kuukauden päivät sekä lisätyt tapahtumat. Tapahtumat näkyvät valitsemallaan värillä merkittynä päivämäärän kohdalla. Käyttäjä voi vaihtaa kuukautta ja valita päivän, jolloin valittu päivämäärä näkyy sivupalkissa kaikkine tapahtumineen. Kuvassa 7 nähdään sovelluksen kalenterinäkymä.



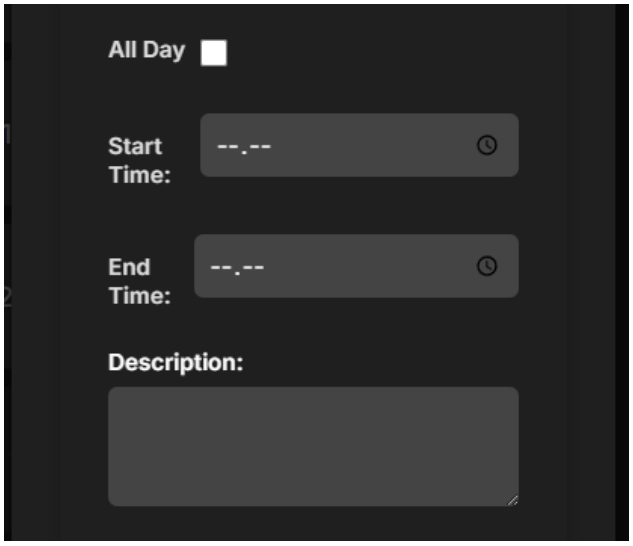
Kuva 7. Kalenterinäkymä.

Tapahtuman lisääminen onnistuu klikkaamalla sivupalkista Add Event -painiketta ja täyttämällä lomakkeen, johon voi kirjata tapahtuman nimen, alku- sekä loppupäivämäärän, haluamansa värin ja halutessaan myös toiston kuten viikoittain, kuukausittain tai vuosittain. Tapahtumaa lisättäessä All Day -vaihtoehto on automaattisesti valittuna, tällöin tapahtuma ei ole sidottu tiettyyn kellonaikaan, vaan se näkyy koko päivän tapahtumana kalenterissa. Tämä on hyödyllistä esimerkiksi lomapäivien, syntymäpäivien tai muiden koko päivän kestävien merkintöjen kirjaamiseen. Kun All Day -vaihtoehtoa ei valita, tapahtumalle voi asettaa tarkan alku- ja loppukellonajan. Kuvassa 8 nähdään tapahtumalomake oletusasetuksilla, kun All Day -vaihtoehto on valittuna. Kuvassa 9 puolestaan näkyy tapahtumalomake ilman All Day -valintaa, jolloin kellonaikakentät ovat käytettävissä.

Lomakkeen suunnittelussa hain inspiraatiota Google Kalenterista, jonka tapahtumalomake toimi lähtökohtana oman rakenteen suunnittelulle. Samankaltainen All Day -toiminto löytyy myös Google Kalenterista.

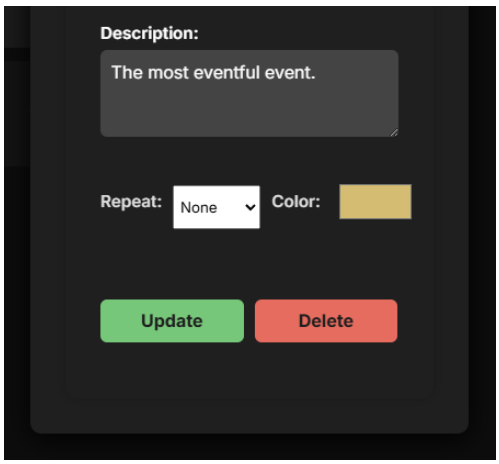


Kuva 8. Tapahtumalomake.



Kuva 9. Tapahtumalomake, kun All Day -vaihtoehto ei ole valittuna.

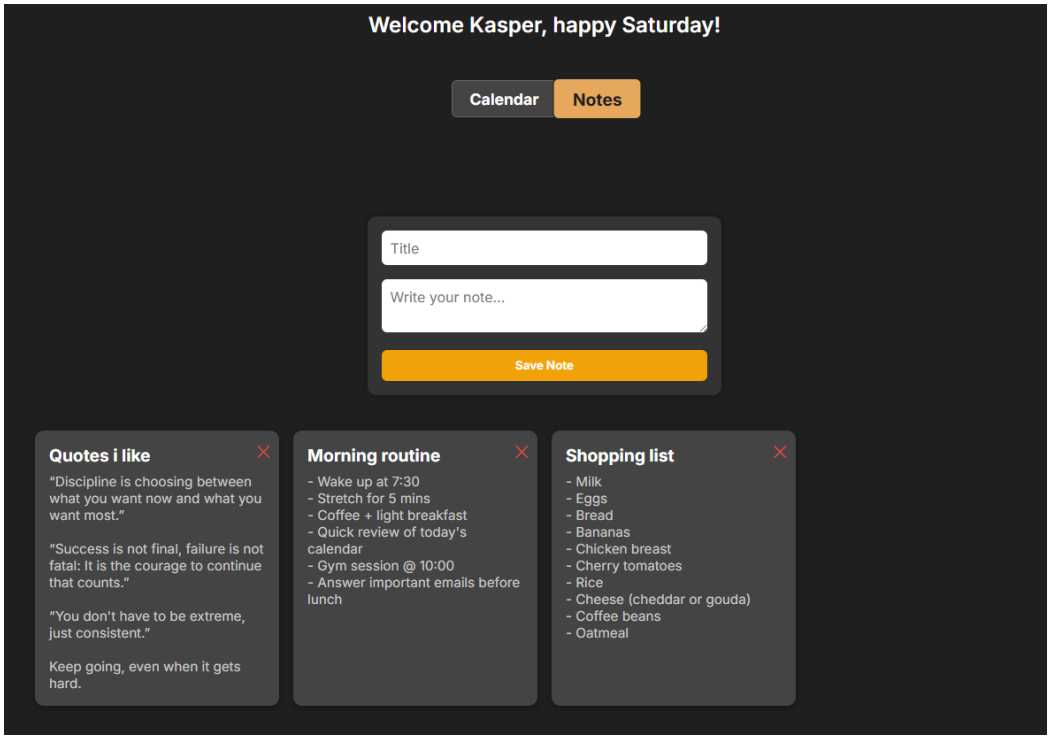
Käyttäjä voi halutessaan myös muokata tai poistaa tapahtumia painamalla sivupalkissa olevaa tapahtumaa, jolloin tapahtumalomake aukeaa ja alhaalta löytyy Update- ja Delete-painikkeet. Kuvassa 10 nähdään tapahtumalomake, jossa on mukana muokkaus- ja poistotoiminnot.



Kuva 10. Tapahtumalomake, jossa mukana Update- ja Delete-painikkeet.

3.3 Muistiinpanonäkymä

Käyttäjä voi siirtyä helposti kalenterinäkymästä muistiinpanonäkymään painamalla ylävalikosta Notes-painiketta. Näkymän tarkoituksena on tarjota käyttäjälle paikka, johon hän voi kirjata ylös tärkeitä asioita, ajatuksia tai muistettavia tehtäviä. Uuden muistiinpanon voi luoda syöttämällä otsikon sekä sisällön lomakkeelle ja painamalla Save Note -painiketta. Jokainen luotu muistiinpano esitetään erillisenä korttina, jossa näkyy otsikko ja sisältö. Muistiinpanot voi helposti poistaa yhdellä napin painalluksella. Kuvassa 11 nähdään kuvankaappaus muistiinpanonäkymästä.

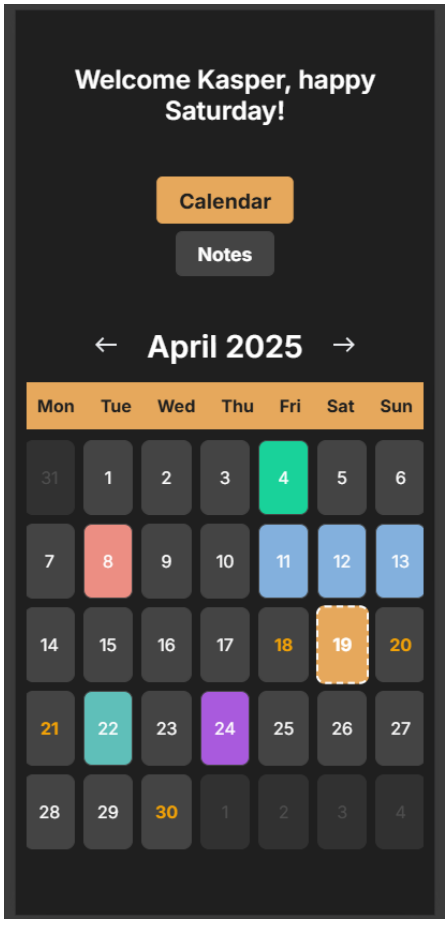


Kuva 11. Kuvankaappaus muistiinpanonäkymästä.

3.4 Mobiililaitenäkymät

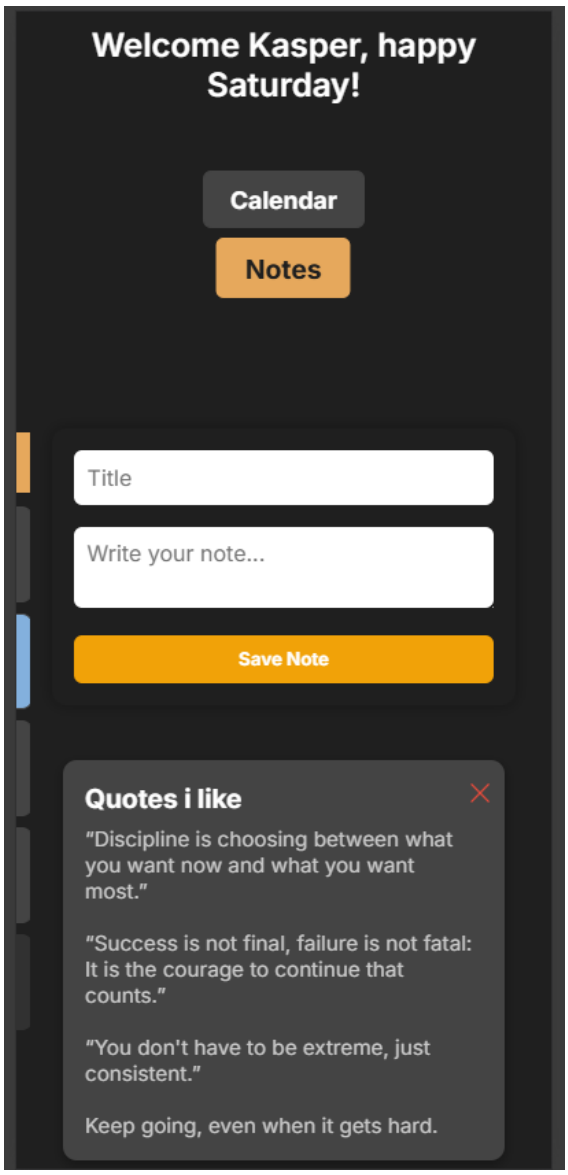
Responsiivisuus oli sovelluksessa tärkeässä roolissa, sillä tällöin kalenteri ja muistiinpanot pysyvät helposti saatavilla mobiililaitteessa ja aina käyttäjän mukana, oli hän sitten missä tahansa.

Kun sovellusta käytetään mobiililaitteella, näkymä skaalautuu automaattisesti näytölle sopivaksi. Esimerkiksi kalenterinäkymässä tapahtumat on sijoitettu siististi päivien alle, ja tapahtuman lisäämiseen tarkoitettu lomake avautuu koko näytön levyisenä, mikä helpottaa kirjoittamista puhelimella. Kuvassa 12 nähdään, miltä kalenterinäkymä näyttää mobiililaitteella.



Kuva 12. Kalenterinäkymä mobiililaitteella (Chrome-kehitystyökalun iPhone 12 Pro -näkymässä).

Kuvassa 13 on muistiinpanonäkymä mobiililaitteella, jossa muistiinpanot asettuvat yksittäisiksi korteiksi pystysuoraan listaan. Näkymien välinen siirtyminen tapahtuu sivun yläpuolella olevista painikkeista.



Kuva 13. Muistiinpanonäkymä mobiililaitteella (Chrome-kehitystyökalun iPhone 12 Pro -näkymässä).

4 SOVELLUKSEN TOTEUTUS

Tässä luvussa tarkastellaan sovelluksen toteutusta teknisestä näkökulmasta. Luvussa esitellään projektin rakenne, teknologien käyttötavat sekä eri osien välinen yhteistoiminta. Koodikatkelmien avulla havainnollistetaan, miten toiminnallisuudet toteutettiin, kuinka tietokantayhteys muodostetaan ja miten käyttäjän kirjautuminen sekä tapahtumien käsittely toimii. Lisäksi käsitellään käyttöliittymän responsiivisuutta eri laitteilla.

4.1 Projektin rakenne

Sovellus on rakennettu noudattaen selkeää jaottelua eri toiminnallisuuksien mukaan, mikä helpottaa sen jatkokehitystä ja ylläpitoa. Projektin juurihakemistossa sijaitseva app.js toimii palvelimen käynnistyspisteenä. Siinä määritellään mm. Expressin käyttö, reititykset, middlewaret sekä näkymäkansion sijainti.

Asennetut Node.js-kirjastot ja riippuvuudet tallentuvat node_modules-kansioon, joka luodaan automaattisesti npm install -komennolla. Projektin metatiedot sekä käynnistysskriptit määritellään package.json-tiedostossa. Tiedosto .gitignore määrittää, mitkä tiedostot jätetään pois versionhallinnasta, kuten node_modules ja .env, joka sisältää arkaluonteiset ympäristömuuttujat kuten tietokantatunnukset. Seuraavaksi esitellään projektin kansiot ja niiden sisältö tarkemmin. Kuvassa 14 näemme myös koko projektin rakenteen Visual Studio Code -editorissa.

- config/
  Tässä kansiossa on db.js tiedosto, joka määrittelee tietokantayhteyden. Yhteys MariaDB-tietokantaan luodaan ympäristömuuttujien avulla, jotka haetaan .env-tiedostosta.

- controllers/
  Kansiossa sijaitsevat sovelluksen logiikasta huolehtivat tiedostot. Käyttäjän tunnistautumiseen, kirjautumiseen ja rekisteröitymiseen liittyvää logiikka löytyy authController.js-tiedostosta. Kaiken logiikan, joka liittyy kalenteritapahtumien käsittelyyn, löytyy eventController.js-tiedostosta, ja muistiinpanojen käsittelyyn liittyvä logiikka löytyy noteController.js-tiedostosta.

- middleware/
  Middleware-kansiossa on yksi tiedosto nimeltä auth.js, josta löytyy JWT-autentikointia varten rakennettu middleware. Tämä tarkistaa käyttäjän evästeessä olevan tokenin ja päästää pyynnön läpi vain, jos käyttäjä on autentikoitu.

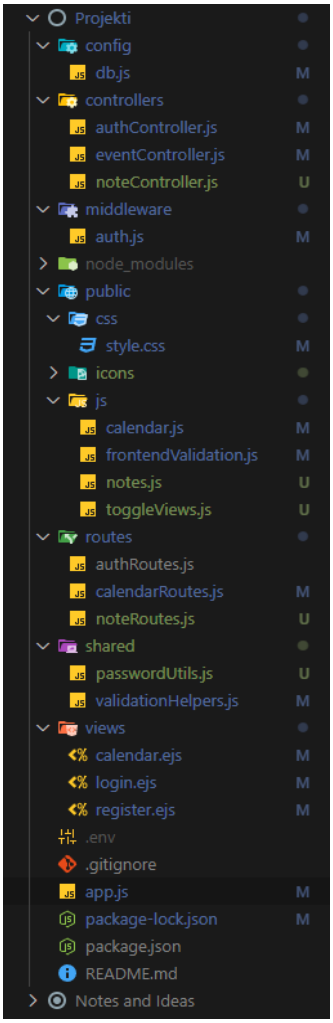
- routes/
  Tässä kansiossa on reititykset, jotka on jaettu niiden käyttötarkoituksen mukaan. authRoutes.js-tiedostossa on rekisteröinnin, kirjautumisen ja uloskirjautumisen reitit. calendarRoutes.js-tiedostossa on kalenteritapahtumien ja noteRoutes.js-tiedostossa on puolestaan muistiinpanojen reitit.

- views/
  Täällä sijaitsevat EJS-tiedostot, joista renderöidään HTML-näkymät. login.ejs ja register.ejs vastaavat kirjautumis- ja rekisteröintisivuista, kun taas calendar.ejs toimii kalenteri- ja muistiinpanonäkymän pohjana.

- public/
  Tämän kansion sisältä löytyy kansio css/ ja sieltä tiedosto style.css, jossa on määritelty käyttöliittymän ulkoasu. Täällä sijaitsee myös toinen kansio nimeltä js/, jossa on tiedostot calen-

dar.js, notes.js ja toggleViews.js, jotka hoitavat front-endin toimintoja sekä lomakevalidointiin liittyvä tiedosto frontendValidation.js. Sovelluksessa käytetyt ikonit löytyvät myös tästä kansiosta.

- shared/
  Tänne on sijoitettu uudelleenkäytettäviä apufunktioita, kuten passwordUtils.js, joka hoitaa salasanan hashauksen ja niiden vertailun. Toinen tiedosto nimeltä validationHelpers.js sisältää validointifunktioita lomakkeiden tarkistukseen.



Kuva 14. Projektin rakenne Visual Studio Code -editorissa.

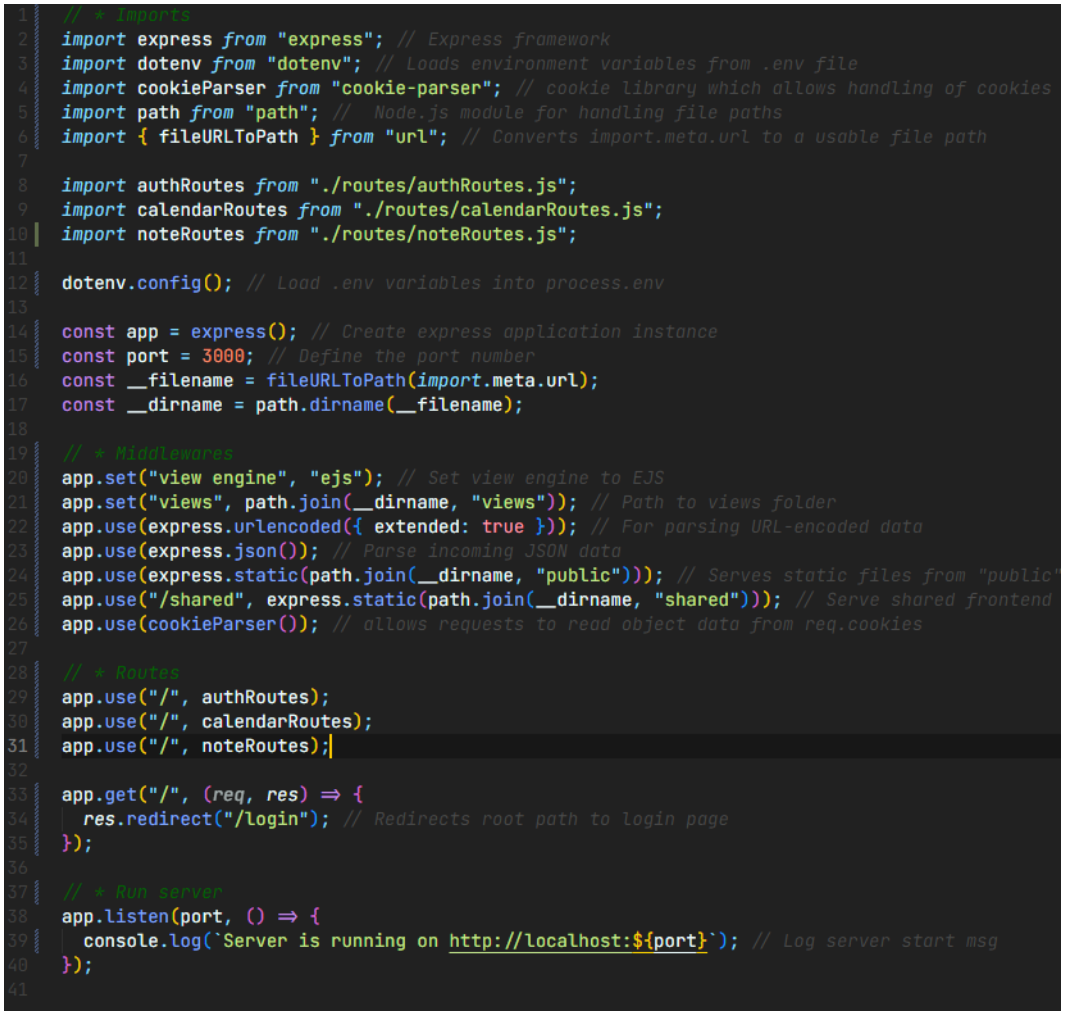
4.2 Backendin rakenne ja toiminta

Tässä luvussa esitellään tarkemmin, miten sovelluksen palvelinpuoli eli backend on toteutettu. Sovelluksen backend vastaa siitä, että käyttäjän selaimesta tulevat pyynnöt käsitellään oikealla tavalla, eli ne ohjataan ensin määritellyille reiteille, tarkistetaan tarvittavien middlewarejen avulla ja käsitellään lopulta kontrollerien kautta. Kontrollerit tekevät tietokantakyselyt ja palauttavat vastaukset takaisin käyttäjälle.

4.2.1 Palvelimen käynnistys ja määritykset

Sovelluksen palvelin käynnistetään tiedostosta app.js, joka toimii koko backendin keskipisteenä. Siinä otetaan käyttöön Express-sovelluskehys ja määritetään perusasetukset, kuten näkymien sijainti (views-kansio), middlewaret sekä staattisten tiedostojen hakemistot. Käynnistyksen yhteydessä määritellään myös, että sovellus kuuntelee porttia 3000 ja itse sovellus käynnistetään komentokehotteesta komennolla ”npx nodemon app.js”.

Middleware on Expressissä välikerros, joka suorittaa jonkin toiminnon saapuvalla HTTP-pyynnöllä ennen kuin se etenee varsinaiseen reittiin. Tässä projektissa on otettu käyttöön esimerkiksi express.json() ja express.urlencoded(), jotka mahdollistavat lomake- ja JSON-datan käsittelyn. Lisäksi cookie-parser-kirjastoa käytetään selaimen evästeiden lukemiseen. Nämä middlewaret ovat tärkeitä esimerkiksi JWT-tunnisteen käsittelyssä ja käyttäjän tunnistautumisessa (Express, n.d.-c). Kuvassa 15 nähdään app.js-tiedosto, jossa määritellään Expressin perusasetukset, käytetyt middlewaret ja sovelluksen reititykset.



```
// * Imports
import express from "express"; // Express framework
import dotenv from "dotenv"; // Loads environment variables from .env file
import cookieParser from "cookie-parser"; // cookie library which allows handling of cookies
import path from "path"; //  Node.js module for handling file paths
import { fileURLToPath } from "url"; // Converts import.meta.url to a usable file path

import authRoutes from "./routes/authRoutes.js";
import calendarRoutes from "./routes/calendarRoutes.js";
import noteRoutes from "./routes/noteRoutes.js";

dotenv.config(); // Load .env variables into process.env

const app = express(); // Create express application instance
const port = 3000; // Define the port number
const __filename = fileURLToPath(import.meta.url);
const __dirname = path.dirname(__filename);

// * Middlewares
app.set("view engine", "ejs"); // Set view engine to EJS
app.set("views", path.join(__dirname, "views")); // Path to views folder
app.use(express.urlencoded({ extended: true })); // For parsing URL-encoded data
app.use(express.json()); // Parse incoming JSON data
app.use(express.static(path.join(__dirname, "public"))); // Serves static files from "public"
app.use("/shared", express.static(path.join(__dirname, "shared"))); // Serve shared frontend
app.use(cookieParser()); // allows requests to read object data from req.cookies

// * Routes
app.use("/", authRoutes);
app.use("/", calendarRoutes);
app.use("/", noteRoutes);

app.get("/", (req, res) => {
  res.redirect("/login"); // Redirects root path to login page
});

// * Run server
app.listen(port, () => {
  console.log(`Server is running on http://localhost:${port}`); // Log server start msg
});
```

Kuva 15. app.js-tiedosto, jossa määritellään Expressin perusasetukset, middlewaret ja reitit.

Tässä projektissa hyödynnettiin myös ECMAScript-moduulijärjestelmää (ESM), joka korvaa perinteisen CommonJS-syntaksin ”require” ja ”module.exports”, modernimmalla import- ja export-syntaksilla. ESM mahdollistaa myös await-komennon käytön tiedostojen päällä ilman erillistä async-funktiota (Ubah, 2024).

4.2.2 Reititys ja REST

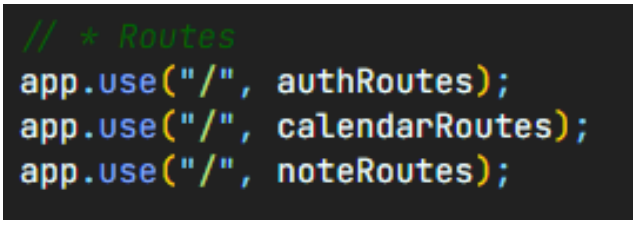
Sovelluksen reititys on toteutettu REST-arkkitehtuurin periaatteiden mukaisesti, joiden yleiset periaatteet on esitelty luvussa 2.1. Sen ajatuksena on, että resurssit kuten tapahtuma tai muistiinpano, käsitellään loogisesti nimetyillä URL-osoitteilla ja standardisoiduilla HTTP-metodeilla (GET, POST, PUT, DELETE). Kun sovellus noudattaa näitä periaatteita, sitä kutsutaan RESTful-sovellukseksi (Gupta, 2025).

Jokainen reitti määrittelee, mitä sovellus tekee tietyllä pyynnöllä. Esimerkiksi GET /event hakee tapahtumat, ja POST /event lisää uuden tapahtuman tietokantaan. Reitit on jaoteltu toiminnallisuuden mukaan omiin tiedostoihinsa routes-kansiossa, kuten authRoutes.js, calendarRoutes.js ja noteRoutes.js. Kuvassa 16 on esitetty HTTP-metodien käyttö calendarRoutes.js-tiedostossa.

```
// * POST event to DB
router.post("/event", authenticateJWT, addEvent);
// * GET all events for logged in user
router.get("/event", authenticateJWT, getEvents);
// * Update event
router.put("/event/:id", authenticateJWT, updateEvent);
// * Delete event
router.delete("/event/:id", authenticateJWT, deleteEvent);
```

Kuva 16. Esimerkki HTTP-metodien käytöstä calendarRoutes.js-tiedostossa.

Edellä mainitut reitit liitetään pääsovellukseen app.js-tiedostossa app.use()-metodin avulla, mikä tekee rakenteesta modulaarisen ja helposti hallittavan. Kuvassa 17 havainnollistetaan, kuinka reitit yhdistetään pääsovellukseen app.js-tiedostossa.



Kuva 17. Reittien yhdistäminen pääsovellukseen app.js-tiedostossa.

Kaikkiin reitteihin on lisätty myös authenticateJWT-middleware, joka tarkistaa käyttäjän tunnistautumisen ennen varsinaisen toiminnallisuuden käsittelyä. Varsinainen logiikka, kuten tietojen tallennus tai muokkaus, on kuitenkin sijoitettu erillisiin tiedostoihin controllers-kansioon. Tätä käsitellään tarkemmin seuraavassa luvussa 4.2.3 Kontrollerit.

4.2.3 Kontrollerit

Kontrollerien tehtävänä on käsitellä reittien kautta saapuvat pyynnöt ja huolehtia tarvittavasta logiikasta, kuten tietokantayhteyksistä, lomakkeen syötteiden validoinnista ja vastauksen palauttamisesta.

Sovelluksessa reitit eivät itse sisällä logiikkaa, vaan ohjaavat pyynnöt eteenpäin oikealle kontrollerille. Kontrollerit sijaitsevat controllers-kansiossa, ja ne on jaettu toiminnallisuuden mukaan käyttäjätunnistautumiseen (authController.js), kalenteritapahtumien hallintaan (eventController.js) ja muistiinpanoihin (noteController.js). Kuvassa 18 on esitetty createNote()-funktio noteController.js-tiedostosta, joka tallentaa uuden muistiinpanon tietokantaan.

```
// * Create a new note
export const createNote = async (req, res) => {
  try {
    const userId = req.user.id; // get user ID from token
    const { title, content } = req.body; // title and content from the form
    const conn = await pool.getConnection(); // opens the DB connection
    await conn.query(
      "INSERT INTO notes (user_id, title, content) VALUES (?, ?, ?)",
      [userId, title, content]
    );
    conn.release(); // closes the DB connection
    res.json({ success: true, message: "Note created" }); // returns a success message
  } catch (err) {
    console.error("Error creating note:", err); // prints error in the console
    res.status(500).send("Server error while creating note"); // returns the error message
  }
};
```

Kuva 18. createNote()-funktio, joka tallentaa uuden muistiinpanon tietokantaan.

Funktio tarkistaa ensin käyttäjän tunnisteen req.user.id:n avulla. Tämä tunniste saadaan purettua JWT-tokenista middleware-vaiheessa. Lomakkeen tiedot (title, content) poimitaan req.body-oliosta. Tietokantayhteys muodostetaan pool.getConnection()-metodilla, ja await conn.query() suorittaa SQL-kyselyn, joka lisää uuden muistiinpanon notes-tauluun. Lopuksi yhteys vapautetaan conn.release()-komennolla ja käyttäjälle palautetaan JSON-muotoinen vastaus. Jos virhe ilmenee, käyttäjälle palautetaan virheilmoitus.

Koodissa käytetään myös async- ja await-rakenteita, jotka ovat osa modernia JavaScriptiä. async määrittää, että funktio voi suorittaa asynkronisia operaatioita, ja await pysäyttää koodin etenemisen, kunnes kyseinen operaatio on valmis, tässä tapauksessa tietokantakysely (Mozilla Developer Network, n.d.-a). Tämä rakenne toistuu kaikissa kontrollereissa, mutta jokainen niistä toimii omalla alueellaan, ja ne tekevät yhteistyötä reittien sekä middlewarejen kanssa.

4.3 Tietokantayhteys

Sovelluksen tietokantayhteys on määritelty erillisessä db.js-tiedostossa, joka sijaitsee config-kansiossa. Tämä tiedosto vastaa yhteyden muodostamisesta MariaDB-tietokantaan käyttäen mariadb -kirjastoa. Yhteys rakennetaan ympäristömuuttujien avulla, ne ladataan .env-tiedostosta. Tämä parantaa tietoturvaa, sillä käyttäjätunnuksia ja salasanoja ei kirjoiteta suoraan koodiin. Kuvassa 19 nähdään, kuinka tietokantayhteys on määritetty.

```
//* Imports

import dotenv from "dotenv";
import mariadb from "mariadb";

dotenv.config(); //Fetch .env file variables

const pool = mariadb.createPool({
  host: process.env.DB_HOST,
  user: process.env.DB_USER,
  password: process.env.DB_PASSWORD,
  database: process.env.DB_NAME,
});

export default pool;
```

Kuva 19. Tietokantayhteyden määrittely db.js-tiedostossa.

Kuvan 19 koodissa luodaan yhteyspooli, jonka avulla sovellus voi hallita useita samanaikaisia yhteyksiä tietokantaan tehokkaasti. Pooli tarjoaa käyttöönsä yksittäisen yhteyden aina tarpeen mukaan, ja yhteys vapautetaan käytön jälkeen.

Tietokantakyselyt, kuten uuden tapahtuman tai muistiinpanon lisääminen, suoritetaan kontrollerien sisällä. Kontrollerit hakevat yhteyden kutsumalla pool.getConnection() ja suorittavat SQL-kyselyt conn.query()-

metodilla. Lopuksi yhteys vapautetaan conn.release() -metodilla. Tämä prosessi on kuvattu tarkemmin luvussa 4.2.3 Kontrollerit, kuvassa 18.

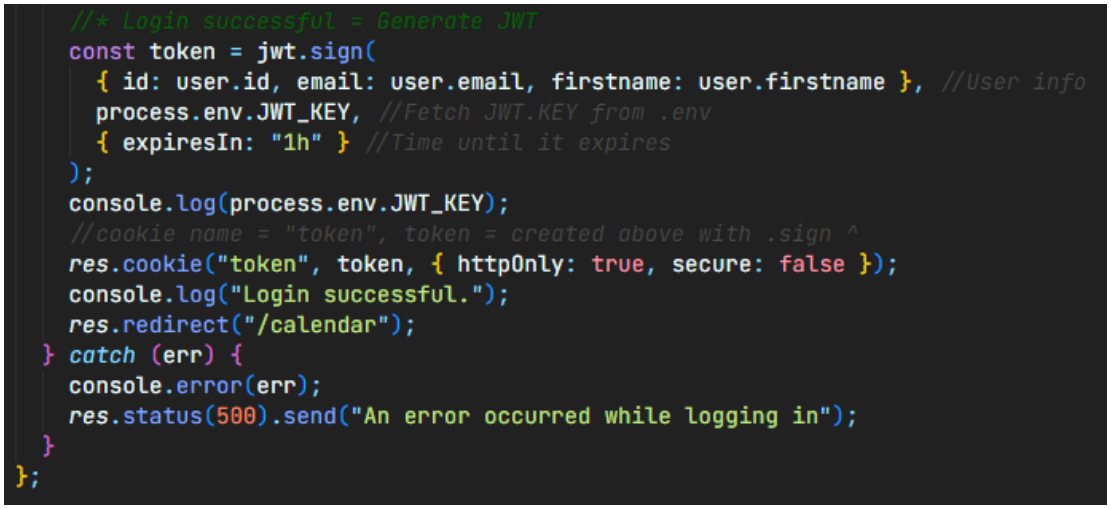
4.4 Autentikointi

Autentikointi eli käyttäjän tunnistaminen on keskeinen osa sovelluksen turvallisuutta. Kun käyttäjä kirjautuu sisään, hänen henkilöllisyytensä varmistetaan ja hänelle annetaan käyttöoikeudet omaan dataansa, kuten kalenteritapahtumiin ja muistiinpanoihin. Tässä sovelluksessa autentikointi on toteutettu JWT-tunnisteiden avulla.

Kun käyttäjä kirjautuu sisään, backend luo hänelle salaisella avaimella allekirjoitetun JWT-tokenin. Tämä tunniste sisältää käyttäjän tunnistetiedot, kuten id, email ja firstname, ja ne lähetetään selaimelle evästeenä. Evästeeseen tallennettu token mahdollistaa sen, että käyttäjä pysyy kirjautuneena, eikä jokaisella sivulatauksella tarvitse tunnistautua uudelleen.

Token tallennetaan evästeeseen res.cookie()-metodilla. Eväste asetetaan httpOnly-tilaan, mikä estää selaimen JavaScriptiä lukemasta tokenia. Tämä suojaa XSS-hyökkäyksiltä, joissa hyökkääjä voisi muuten varastaa käyttäjän kirjautumistiedot (OWASP, n.d.-a). Kehitysympäristössä evästeelle on asetettu secure: false, jotta se toimii myös ilman HTTPS-yhteyttä. Tuotantoympäristössä tämä asetus kannattaa muuttaa secure: true, jolloin eväste lähetetään vain suojatun yhteyden kautta.

Kuvassa 20 nähdään authController.js-tiedostosta koodikappale, jossa käyttäjälle luodaan JWT-tunnus onnistuneen kirjautumisen jälkeen.



```
    //* Login successful = Generate JWT
    const token = jwt.sign(
      { id: user.id, email: user.email, firstname: user.firstname }, //User info
      process.env.JWT_KEY, //Fetch JWT.KEY from .env
      { expiresIn: "1h" } //Time until it expires
    );
    console.log(process.env.JWT_KEY);
    //cookie name = "token", token = created above with .sign ^
    res.cookie("token", token, { httpOnly: true, secure: false });
    console.log("Login successful.");
    res.redirect("/calendar");
  } catch (err) {
    console.error(err);
    res.status(500).send("An error occurred while logging in");
  }
};
```

Kuva 20. JWT-tokenin luonti onnistuneen kirjautumisen jälkeen (auth-Controller.js).

Kaikki reitit, jotka vaativat tunnistautumista, käyttävät authentica-teJWT-middlewarea, joka tarkistaa selaimen lähettämän tokenin oikeellisuuden. Tämä middleware toimii ikään kuin portinvartijana, joka joko päästää pyynnön eteenpäin tai ohjaa käyttäjän takaisin kirjautumissivulle, jos tunnistautuminen epäonnistuu. Kuvassa 21 on esitetty authenticateJWT-middleware, jossa tarkistetaan käyttäjän JWT-tunniste.

```
import jwt from "jsonwebtoken"; // Fetch JWT library

export const authenticateJWT = (req, res, next) => {
  const token = req.cookies.token; // Get user token from cookies

  // If no token found, redirect ⟶ /login
  if (!token) {
    return res.redirect("/login");
  }

  try {
    const user = jwt.verify(token, process.env.JWT_KEY); // Verify that token is valid and not expired
    req.user = user; // Adds user data to req.user, so it can be used
    next(); // Move to next middleware/route
  } catch (err) {
    console.error(err);
    res.redirect("/login");
  }
};
```

Kuva 21. JWT-tunnisteen tarkistus authenticateJWT-middlewarella (auth.js).

Kyseisellä ratkaisulla käyttäjä voidaan tunnistaa turvallisesti jokaisella pyynnöllä ilman perinteistä istunnonhallintaa.

4.5 Frontend ja backend -yhteistyö

Sovelluksen frontend- ja backend-osat tekevät yhteistyötä. Käyttäjän selaimessa tekemät toiminnot, kuten tapahtuman lisääminen tai muistiinpanon poistaminen, käynnistävät JavaScriptin kautta HTTP-pyynnön palvelimelle. Tiedonsiirrossa käytetään fetch-metodia, joka hyödyntää AJAX-tekniikkaa (Asynchronous JavaScript and XML). AJAX mahdollistaa dynaamisen tiedonsiirron, mikä tarkoittaa, että sivua ei tarvitsee ladata uudelleen, tämä tekee sovelluksesta nopeamman ja käyttäjäystävällisemmän (Mozilla Developer Network, n.d.-d).

Kun käyttäjä esimerkiksi lisää uuden tapahtuman kalenteriin, frontendissä oleva calendar.js-tiedosto käsittelee lomakkeen lähetyksen ja tekee POST-pyynnön backendille. Kuvassa 22 havainnollistetaan, kuinka tapahtuma lähetetään backendille fetch-metodilla.

```
try {
  const res = await fetch("/event", {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
    },
    body: JSON.stringify(eventData),
  });
```

Kuva 22. Tapahtuman lähetys palvelimelle fetch-metodilla calendar.js-tiedostossa.

Pyyntö etenee tästä backendin reitille POST /event, joka ohjaa sen eventController.js-tiedostossa sijaitsevaan kontrolleriin. Kontrolleri tallentaa tapahtuman tietokantaan ja palauttaa sitten vastauksen JSON-muodossa. Frontend käsittelee vastauksen ja päivittää kalenterinäkymän kutsumalla fetchAndRenderEvents()-funktiota, joka hakee ajantasaiset tapahtumat uudelleen ilman sivunpäivitystä.

Samanlainen logiikka toimii myös muistiinpanojen poistamisessa. Kun käyttäjä painaa Delete-painiketta, notes.js lähettää DELETE-pyynnön palvelimelle. Kuvassa 23 havainnollistetaan, kuinka muistiinpanon poistaminen toteutetaan fetch-pyynnöllä.

Kuva 23. Fetch-pyyntö muistiinpanon poistamiseksi notes.js-tiedostossa.

Pyyntö kulkee reitin DELETE /notes/:id kautta noteController.js-tiedostoon, joka poistaa muistiinpanon tietokannasta ja palauttaa onnistumisviestin. Frontend reagoi tähän poistamalla muistiinpanon näkymästä. Tämä rakenne tekee sovelluksesta joustavan ja nopean käyttää ilman sivun uudelleenlatauksia.

4.6 Kalenterin toteutus

Sovelluksen kalenterikomponentti sijaitsee tiedostossa /public/js/calendar.js. Se luo kuukausinäkymän päivistä, hakee kirjautuneen käyttäjän tapahtumat, laajentaa toistuvat tapahtumat näkyviksi ja tarjoaa modaalin tapahtumien lisäämistä, muokkaamista ja poistamista varten. Tämä luku kuvaa kalenterin tärkeimmät tekniset ratkaisut.

4.6.1 Kalenteriruudukon generointi

Tärkein osa itse kalenteria on funktio updateCalendar(), joka rakentaa jokaisella kutsulla uuden kuukausiruudukon. Funktio laskee kuukauden ensimmäisen viikonpäivän sekä kuukauden päivien kokonaismäärän. Jotta kalenteri pysyisi tasaisena ja alkaisi aina maanantaista ja päättyy

sunnuntaihin, ruudukon alkuun sekä loppuun lisätään tarvittavat edellisen ja seuraavan kuukauden päivät, jotka näkyvät haalistuneina. Päivät generoidaan JavaScriptin createElement()-metodin avulla, ja niille annetaan tarvittavat luokat kuten "day", "faded" tai "today". Kuvassa 24 havainnollistetaan, miten päivien selvittäminen ja haalistuneiden päivien lisääminen kalenteriruudukon alkuun toteutetaan calendar.js-tiedostossa.

```
// Figure out the first day of the month
const firstDayOfMonth = new Date(year, month, 1).getDay(); //0-6 = Sun-Sat
// Figure out the length of the month
const totalDaysOfMonth = new Date(year, month + 1, 0).getDate();

// Calculate days of the previous months boxes that we need before the first day of the month
const daysFromPrevMonth = firstDayOfMonth === 0 ? 6 : firstDayOfMonth - 1;
const lastDayOfPrevMonth = new Date(year, month, 0).getDate();

// In the loop we add the previous months days to the start of the calendar
for (let i = daysFromPrevMonth - 1; i >= 0; i--) {
  const fadedDays = document.createElement("div");
  fadedDays.classList.add("day", "faded"); // Add css class "day" and "faded"
  fadedDays.textContent = lastDayOfPrevMonth - i; //  Fill with correct dates ...29, 30, 31...
  calendar.appendChild(fadedDays); // Add the faded day box to the calendar grid
}
```

Kuva 24. Päivien selvittäminen ja haalistuneiden päivien lisäys kalenterin alkuun (calendar.js).

Yllä olevan koodiesimerkin perään täytetään kalenteri valitun kuukauden päivillä, minkä jälkeen lisätään haalistuneet päivät kuukauden loppuun käyttäen samaa periaatetta kuten kuukautta edeltäviin haalistuneisiin päiviin.

4.6.2 Tapahtumien haku

Kun kalenterinäkymä avataan tai käyttäjä vaihtaa kuukautta, kutsuu sovellus fetchAndRenderEvents()-funktiota. Tämä funktio hakee kirjautuneen käyttäjän tapahtumat palvelimen reitiltä GET /event, joka palauttaa tapahtumat JSON-muodossa tietokannasta.

Kuva 25 havainnollistaa, miten fetchAndRenderEvents()-funktio aloittaa hakuprosessin. Tapahtumat vastaanotetaan fetch-pyynnön avulla, minkä jälkeen tapahtumat käsitellään yksitellen forEach-silmukassa.

```
// * Function to fetch and render the events
async function fetchAndRenderEvents() {
  try {
    const res = await fetch("/event"); // fetch all user events
    const events = await res.json(); // events returned in JSON

    const visibleYear = year;
    const visibleMonth = month;

    const instanceOfEvents = [];

    // Loop through each event
    events.forEach((event) => {
```

Kuva 25. Tapahtumien haku fetchAndRenderEvents()-funktion alussa, ja tapahtumien läpikäynti events.forEach(event)-silmukassa.

Jos haetulla tapahtumalla on eri alku- ja loppupäivämäärä, jaetaan se useiksi yksittäisiksi päiviksi, jolloin se näkyy kalenterissa jokaisena päivänä tapahtuman keston ajan. Mikäli tapahtumalle on määritelty repeat-kenttä (esimerkiksi weekly, monthly tai yearly), käytetään repeatEvent-apufunktiota, joka luo näkyviin tarvittavat toistuvat esiintymät kalenterinäkymän sisällä. Toistuvia kopioita ei tallenneta erikseen tietokantaan,

vaan ne luodaan vain esittämistä varten, mikä pitää tietokannan datan kevyenä ja rakenteen yksinkertaisena.

Lopuksi luodut tapahtumaoliot tallennetaan allEvents-taulukkoon, joka on asetettu globaaliksi muuttujaksi. Tätä käytetään kalenterin päivien merkintään ja valitun päivän tapahtumien näyttämiseen sivupalkissa. Tämän jälkeen kalenteri päivitetään kutsumalla updateCalendar()-funktiota, joka piirtää näkymän uudelleen tapahtumien kanssa.

4.6.3 Päivän valinta ja tapahtumien listaus

Sovelluksessa käyttäjän vuorovaikutus perustuu suurelta osin tapahtumakuuntelijoihin (event listener). Niiden avulla sovellus reagoi erilaisiin toimiin, kuten lomakkeen lähettämiseen, painikkeen klikkaamiseen tai kalenteripäivän valintaan.

Kun päivää klikkaa kalenterissa, sovellus tunnistaa käyttäjän valinnan tapahtumakuuntelijan avulla. Se tarkistaa, onko klikattu elementti varsinainen päivä (eli div, jolla on luokka .day), ja muodostaa siitä päivämäärämerkkijonon muodossa YYYY-MM-DD. Tämän jälkeen kutsutaan funktiota selectDate(dateStr). Kuvassa 26 havainnollistetaan tapahtumakuuntelija, joka muodostaa klikatusta päivästä päivämäärämerkkijonon ja kutsuu sitten selectDate()-funktiota.

```
// * On day click, show the day in sidebar
calendar.addEventListener("click", function (e) {
  const dayElement = e.target.closest(".day");
  if (dayElement && !dayElement.classList.contains("faded")) {
    const clickedDay = dayElement.textContent.trim();
    const paddedMonth = String(month + 1).padStart(2, "0");
    const paddedDay = String(clickedDay).padStart(2, "0");
    const dateStr = `${year}-${paddedMonth}-${paddedDay}`;
    selectDate(dateStr);
  }
});
```

Kuva 26. Tapahtumakuuntelija, joka muodostaa valitusta päivästä merkkijonon ja kutsuu selectDate()-funktiota.

Funktio selectDate() vastaa valitun päivän korostamisesta kalenterissa ja päivämäärän näyttämisestä sivupalkin otsikossa. Tämän jälkeen kutsutaan showEventsForDate()-funktiota, joka suodattaa allEvents-taulukosta valittuun päivään kuuluvat tapahtumat ja järjestää ne sivupalkkiin. Ensin näytetään koko päivän tapahtumat ja sitten kellonaikaan sidotut.

Kun käyttäjä klikkaa yksittäistä tapahtumaa sivupalkissa, se avataan muokkaustilassa. Tämäkin on toteutettu tapahtumakuuntelijan avulla. Lomake täytetään klikatun tapahtuman tiedoilla, ja käyttäjä voi päivittää tai poistaa sen. Näkymä säilyy valitussa päivässä myös muokkauksen jälkeen, sillä selectedDate-muuttujaa ei nollata päivityksen yhteydessä.

4.6.4 Kuukausinavigointi

Kalenterinäkymässä käyttäjä voi siirtyä edelliseen tai seuraavaan kuukauteen käyttämällä nuolinäppäimiä, jotka sijaitsevat kuukausinimen

molemmin puolin. Nämä painikkeet on määritelty EJS-tiedostossa elementeiksi #prev-month ja #next-month, ja niiden toiminta toteutetaan tapahtumakuuntelijoiden avulla.

Kun nuolta painetaan, sovellus tarkistaa nykyisen kuukauden ja päivittää month- ja year-muuttujia tilanteen mukaan. Siirtyminen joulukuusta eteenpäin kasvattaa vuosilukua ja nollaa kuukauden tammikuuksi (0), ja vastaavasti tammikuusta taaksepäin siirtyminen vähentää vuosilukua ja asettaa kuukaudeksi joulukuun (11).

Kuvassa 27 nähdään, miten edelliseen kuukauteen siirtyminen toteutetaan tapahtumakuuntelijalla. Kuukauden ja vuoden päivityksen jälkeen kutsutaan fetchAndRenderEvents()-funktiota, joka hakee uudet tapahtumat valitulle kuukaudelle. Tämä funktio päivittää lopuksi näkymän kutsumalla updateCalendar()-funktiota, jolloin kalenteriruudukko piirretään uudelleen ajantasaisilla tapahtumatiedoilla. Näin näkymä päivittyy dynaamisesti ja tapahtumamerkinnät pysyvät ajantasaisina.

```
// * Eventlistener for button to change to previous month
prevButton.addEventListener("click", () => {
  if (month === 0) {
    month = 11; // if its month 0, go back to December of last year
    year--;
  } else {
    // else just go back one month andd dont change the year
    month--;
  }
  fetchAndRenderEvents(); // Refresh the calendar
});
```

Kuva 27. Edelliseen kuukauteen siirtymisen tapahtumakuuntelija.

4.6.5 Tapahtumalomake

Tapahtumien lisääminen, muokkaaminen ja poistaminen tapahtuvat kalenterin yhteydessä avautuvan lomakkeen avulla. Tämä tapahtumalomake on toteutettu modaalina (#event-modal), joka on oletuksena piilotettu. Se ilmestyy näkyviin, kun käyttäjä painaa Add Event -painiketta tai klikkaa yksittäistä tapahtumaa sivupalkista.

Kuvassa 28 on esitetty koodiesimerkillä, miten Add Event -painike avaa lomakkeen. Ennen modaalin näyttämistä lomake alustetaan, kentät tyhjennetään, valittu päivä esitäytetään, All Day -valinta asetetaan oletukseksi ja turhat painikkeet piilotetaan. Tämän jälkeen modaali näytetään käyttäjälle.

```
addEventBtn.addEventListener("click", () => {
  if (!selectedDate) return;

  eventForm.reset(); // Reset form
  document.getElementById("event-id").value = "";
  document.getElementById("edit-buttons").classList.add("hidden"); // Hides delete and update
  eventForm.querySelector("button[type='submit']").classList.remove("hidden"); // Reveal Save

  // Prefill selected date
  document.getElementById("event-start-date").value = selectedDate;
  document.getElementById("event-end-date").value = selectedDate;

  // Set All Day checked and hide time inputs
  const allDayCheckbox = document.getElementById("event-all-day");
  allDayCheckbox.checked = true;
  document.getElementById("time-fields").style.display = "none";

  // Show the modal
  modal.classList.remove("hidden");
});
```

Kuva 28. Add event -painikkeen tapahtumakuuntelija.

Lomakkeen lähettämistä kuunnellaan myös tapahtumakuuntelijalla. Kun tätä painetaan, tiedot pakataan eventData-olioon jonka jälkeen ne lähetetään palvelimelle POST-pyyntönä. Jos kyseessä on muokkaus, mukana kulkee piilotettu event-id-kenttä, jotta tiedetään, mitä tapahtumaa

muokataan. Onnistuneen toiminnon jälkeen kalenteri päivitetään kutsumalla fetchAndRenderEvents() ja säilyttämällä valittu päivä (selectedDate) aktiivisena.

4.7 Muistiinpanojen toteutus ja näkymien vaihto

Muistiinpanot on toteutettu teknisesti omana näkymänään, joka sijaitsee samassa HTML-sivussa kalenterin kanssa. Käyttäjä voi vaihtaa näkymien välillä käyttöliittymän yläpalkissa olevilla painikkeilla, ja näkymän vaihto tapahtuu toggleViews.js-tiedostossa. Kun käyttäjä painaa Notes-painiketta, kalenterinäkymä liukuu pois ja muistiinpanot ilmestyvät näkyviin. Tämän toiminnallisuus toteutettiin JavaScriptillä ja animaatio CSS:llä. Kuvassa 29 esitetään, kuinka näkymän vaihto kalenterin ja muistiinpanojen välillä on toteutettu tapahtumakuuntelijoiden avulla luokkia vaihtamalla.

```
// * Get the buttons for switching between calendar and notes
const calendarBtn = document.getElementById("calendar-view-btn");
const notesBtn = document.getElementById("notes-view-btn");
// Get the wrapper which contains both view
const sliderWrapper = document.querySelector(".view-slider");

// Calendar button
calendarBtn.addEventListener("click", () => {
  sliderWrapper.classList.remove("view-notes"); // show calendar
  calendarBtn.classList.add("active"); // add calendar button as active
  notesBtn.classList.remove("active"); // remove notes button from active
});

// Notes button
notesBtn.addEventListener("click", () => {
  sliderWrapper.classList.add("view-notes"); // show notes
  notesBtn.classList.add("active");
  calendarBtn.classList.remove("active");
});
```

Kuva 29. toggleViews.js ja näkymienvaihdon toteutus.

Muistiinpanojen toiminnallisuus on toteutettu notes.js-tiedostossa. Kun käyttäjä lähettää uuden muistiinpanon lomakkeella, tiedot lähetetään POST-pyynnöllä palvelimelle. Onnistuneen tallennuksen jälkeen kutsutaan loadNotes()-funktio, joka hakee kaikki muistiinpanot GET /notes -reitiltä ja luo niistä elementtejä JavaScriptin avulla. Kuvassa 30 nähdään, miten loadNotes()-funktio käy läpi palvelimelta haetut muistiinpanot ja rakentaa niistä kortit käyttöliittymään.

```
// * Fetch and display notes
async function loadNotes() {
  const res = await fetch("notes");
  const notes = await res.json();

  notesContainer.innerHTML = ""; // clear existing

  // Loop through each note and create a notecard
  notes.forEach((note) => {
    const noteCard = document.createElement("div");
    noteCard.classList.add("note-card");

    // Delete button (top-right corner)
    const deleteBtn = document.createElement("button");
    deleteBtn.textContent = "×";
    deleteBtn.classList.add("delete-note");
    deleteBtn.dataset.id = note.id; // store note id
    noteCard.appendChild(deleteBtn);

    // Create and add Title
    const title = document.createElement("h3");
    title.textContent = note.title;
    noteCard.appendChild(title);

    // Create and add Content
    const content = document.createElement("p");
    content.textContent = note.content;
    noteCard.appendChild(content);

    // Add note card to notesContainer
    notesContainer.appendChild(noteCard);
  });
}
```

Kuva 30. loadNotes()-funktio.

4.8 Responsiivisuus ja käyttöliittymä

Kalenteri- ja muistiinpanosovelluksen käyttöliittymä suunniteltiin alusta alkaen toimimaan sekä tietokoneilla että mobiililaitteilla. Responsiivisuus toteutettiin käyttämällä skaalautuvia yksiköitä, kuten rem, ja hyödyntämällä @media-kyselyitä style.css-tiedostossa. Tämän ansiosta sovelluksen rakenne mukautuu automaattisesti näytön koon mukaan.

Sovelluksen visuaalinen ilme määriteltiin root-valitsimella, jossa asetettiin yleiset värimuuttujat, kuten tekstin ja taustan väri sekä korostusvärit. Näitä voidaan käyttää tyylisäännöissä esimerkiksi muodossa background-color: var(--background). Tämä tekee värien hallinnasta keskitettyä ja helpottaa mahdollisia teemamuutoksia myöhemmin. Kuvassa 31 on esitetty root-valitsin, jossa sovelluksen teema-asetukset määritellään.

```
:root {
  --text: #ffffff;
  --background: #1f1f1f;
  --primary: #e6a85c;
  --secondary: #444;
}
```

Kuva 31. Teema-asetukset root-valitsimessa.

Kalenterinäkymä on rakennettu ruudukoksi, jossa viikonpäivät jakautuvat seitsemään sarakkeeseen. Kalenterin ja sivupalkin asettelu perustuu flex-rakenteeseen. Kun näyttö pienenee alle 768px, joka on web-kehityksessä yleisesti käytetty katkaisupiste, elementtien asettelu muuttuu automaattisesti pystysuuntaiseksi flex-direction: column -määrityksellä, mikä mahdollistaa luettavan ja käyttökelpoisen näkymän myös mobiililaitteella (w3schools, n.d.). Melkein kaikki style.css-tiedoston px-arvot

on muunnettu rem-arvoiksi CSS Unit Converterilla, parantamaan responsiivisuutta (CSS Unit Converter n.d.). Kuvassa 32 nähdään, miten @media-query on toteutettu.

```
@media (max-width: 768px) {
  .container {
    padding: 0 0.625rem;
  }

  .calendar-layout {
    flex-direction: column;
    align-items: stretch;
  }
```

Kuva 32. @media-query muuttaa kalenterin asettelua siirtämällä sivupalkin kalenterin alapuolelle, kun näyttö on alle 768px.

5 YHTEENVETO

Tässä opinnäytetyössä toteutettiin selainpohjainen kalenteri- ja muistiinpanosovellus. Tavoitteena oli rakentaa sovellus, joka toimii joustavasti eri laitteilla ja tarjoaa mahdollisuuden kirjautumiseen, tapahtumien hallintaan ja henkilökohtaisten muistiinpanojen tallentamiseen tietokantaan.

Työssä päädyttiin käyttämään JavaScript-pohjaisia ratkaisuja, kuten Node.js:ää ja Expressiä, koska ne tukevat tehokkaasti modernin websovelluksen rakentamista. Frontendissä hyödynnettiin HTML:ää, CSS:ää ja JavaScriptiä, ja näkymien hallintaan käytettiin EJS:ää. Tietokantana toimi MariaDB.

Kaikki tavoitteet saavutettiin hyvin ja työn lopputuloksena syntyi toimiva, turvallinen ja laajennettavaksi suunniteltu web-sovellus. Jatkokehityksen kannalta sovellukseen olisi mahdollista lisätä uusia ominaisuuksia, kuten muistutuksia, teemoja tai jopa kalenterisynkronointia muiden palveluiden kanssa.

Toteutusprosessin aikana karttui paljon käytännön kokemusta modernin web-sovelluksen rakentamisesta alusta alkaen. Työssä opin mm. reitityksestä, lomakevalidoinneista, autentikoinnista ja frontendin dynaamisesta päivittämisestä, joista osa oli minulle aivan uusia asioita. Kokonaisuutena projekti vahvisti osaamistani merkittävästi ja antoi varmuutta tulevaisuuden projekteja varten.

LÄHTEET

Aaron Bond. (n.d.). Aaron Bond – Projects. Noudettu 21.4.2025 osoitteesta https://aaronbond.co.uk/

Auth0. (n.d.). JSON Web Tokens (JWT). Noudettu 19.4.2025 osoitteesta https://auth0.com/docs/secure/tokens/json-web-tokens

bcrypt.js. (n.d.). bcrypt.js GitHub repository. Noudettu 19.4.2025 osoitteesta https://github.com/dcodeIO/bcrypt.js

CCS Unit Converter. (n.d.). CSS Unit Converter. Noudettu 25.4.2025 osoitteesta https://cssunitconverter.vercel.app/

EJS. (n.d.). Embedded JavaScript templates. Noudettu 20.4.2025 osoitteesta https://ejs.co/

Express. (n.d.-a). Express cookie-parser middleware. Noudettu 19.4.2025 osoitteesta https://expressjs.com/en/resources/middleware/cookie-parser.html

Express. (n.d.-b). Node.js web application framework. Noudettu 18.4.2025 osoitteesta https://expressjs.com/

Express. (n.d.-c). Using middleware. Noudettu 24.4.2025 osoitteesta https://expressjs.com/en/guide/using-middleware.html

GitHub. (n.d.). GitHub Docs. Noudettu 19.4.2025 osoitteesta https://docs.github.com/en

Lokesh Gupta. (1. huhtikuuta 2025). What is REST? Noudettu 22.04.2025 osoitteesta https://restfulapi.net/

MariaDB Foundation. (n.d.-a). MariaDB - Open source database. Noudettu 18.4.2025 osoitteesta https://mariadb.org/

MariaDB Foundation. (n.d.-b). MariaDB Node.js connector. Noudettu 18.4.2025 osoitteesta https://mariadb.com/kb/en/nodejs-connector/

Motdotla. (n.d.). dotenv-npm. Noudettu 18.4.2025 osoitteesta https://www.npmjs.com/package/dotenv

Mozilla Developer Network. (n.d.-a) async function – JavaScript | MDN. Noudettu 23.4.2025 osoitteesta https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async_function

Mozilla Developer Network. (n.d.-b). CSS media queries. Noudettu 20.4.2025 osoitteesta https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_media_queries

Mozilla Developer Network. (n.d.-c). REST. Noudettu 18.4.2025 osoitteesta https://developer.mozilla.org/en-US/docs/Glossary/REST

Mozilla Developer Network. (n.d.-d). Using the Fetch API. Noudettu 24.4.2025 osoitteesta https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch

Node.js. (n.d.). Introduction to Node.js. Noudettu 18.4.2025 osoitteesta https://nodejs.org/en/learn/getting-started/introduction-to-nodejs

OWASP. (n.d.-a). OWASP Foundation HttpOnly. Noudettu 25.4.2025 osoitteesta https://owasp.org/www-community/HttpOnly

OWASP. (n.d.-b) Password Storage – OWASP Cheat Sheet Series. Noudettu 20.4.2025 osoitteesta https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html

Prettier. (n.d.) Prettier – Opinionated Code Formatter. Noudettu 21.4.2025 osoitteesta https://prettier.io/

Remy Sharp. (n.d.). nodemon-npm. Noudettu 20.4.2025 osoitteesta https://www.npmjs.com/package/nodemon

Ubah Kingsley. (6. kesäkuuta 2024). CommonJS vs. ES modules in Node.js. Noudettu 22.4.2025 osoitteesta https://blog.logrocket.com/commonjs-vs-es-modules-node-js/#commonjs-vs-es-modules-syntax

Visual Studio Code. (n.d.). Visual Studio Code. Noudettu 18.4.2025 osoitteesta https://code.visualstudio.com/

w3schools. (n.d.). Responsive Web Design Media Queries. Noudettu 25.4.2025 osoitteesta https://www.w3schools.com/css/css_rwd_mediaqueries.asp