LAB University of Applied Sciences

# Tehtävienhallintajärjestelmän toteutus

LAB-ammattikorkeakoulu
Insinööri (AMK)
2022
Kalle Louhimies

**Tiivistelmä**


| Tekijä(t) | Julkaisun laji | Valmistumisaika |
|---|---|---|
| Louhimies, Kalle | Opinnäytetyö, AMK | 2022 |
| | Sivumäärä | |
| | 43 | |


Työn nimi

**Tehtävienhallintajärjestelmän toteutus**

Tutkinto ja koulutusala

Insinööri (AMK), tieto- ja viestintätekniikka, ohjelmistotekniikka

Toimeksiantajan nimi, titteli ja organisaatio (jos opinnäytetyöllä on toimeksiantaja)

Teemu Parkkinen, CTO, Kuuki Tech Oy


Tiivistelmä.

Työn tavoitteena oli toteuttaa tehtävienhallintajärjestelmä web-sovelluksena. Työn toimeksiantaja oli Kuuki Tech Oy. Päätavoitteena oli luoda sovellus, jolla voi luoda ja muokata sekä projekteja että tehtäviä ajasta ja paikasta riippumatta. Muina tavoitteina oli tehdä sovelluksen käyttöliittymä käyttäjäystävälliseksi ja käsitellä käyttäjien tietoja tietoturvallisesti.

Tehtävienhallintajärjestelmälle luotiin MariaDB-tietokanta. Järjestelmän käyttöliittymä toteutettiin Vue.js-kehyksellä. Käyttöliittymän yhdistämiseksi tietokantaan luotiin Node.js-taustajärjestelmä Express-kehyksellä.

Tuloksena oli toimiva sovellus. Myös muut tavoitteet saavutettiin ja tehtävienhallintajärjestelmään lisättiin sen käyttämistä helpottavia toiminnallisuuksia. Jatkokehitysideoita kartoitettiin työn aikana.

Tehtävienhallintajärjestelmä soveltuu esimerkiksi ohjelmointiprojektien ja niihin kuuluvien tehtävien hallintaan. Sitä voi myös käyttää kalenteri- ja muistiosovelluksena.



Asiasanat

tehtävienhallintajärjestelmä, SPA, web-sovellus, full stack

**Abstract**


| Author(s) | Type of Publication | Published |
|---|---|---|
| Louhimies, Kalle | Thesis, UAS | 2022 |
| | Number of Pages | |
| | 43 | |


Title of Publication

**Developing a Task Management Application**

Degree and field of study

Bachelor of Engineering, Information Technology, Software Engineering

Name, title, and organisation of the client (if the thesis work is commissioned by another party)

Teemu Parkkinen, CTO, Kuuki Tech Oy


Abstract

The aim of this thesis was to develop a task management application as a modern web application for Kuuki Tech Oy. The primary objective was to create an application with which the user can create, read, update, and delete projects and tasks. The secondary objectives were to implement a user-friendly user interface and to manage user data in a cybersecure way.

The task management application's frontend was built with the Vue.js framework. The backend was developed with Node.js and the Express framework. The application's database was implemented with MariaDB.

Both the primary and secondary objectives were achieved, resulting in a functioning web application for managing projects and tasks. Additional functionalities were added to better the user experience. Ideas for further development of the system were documented.

The created task management application is useful for managing software development projects and tasks and can also be used as a calendar and memo application.



Keywords

task management application, SPA, web application, full stack

Sisällys

## Termit ja lyhenteet

| | |
|---|---|
| API | Ohjelmistorajapinta. Yhteys ohjelmien tai ohjelman ja komponentin välillä. |
| bcrypt | Salausväliohjelmisto muun muassa salasanojen salaamiseen. |
| CSS | Tyylisivu dokumenttien tyylittelyyn. |
| Express | Ohjelmistokehys taustajärjestelmien ja API:en toteuttamiseen. |
| Full stack | Web-sovelluksen kaikki tasot; käyttöliittymä, taustajärjestelmä ja tietokanta. |
| HTML | Kieli verkkosivun rakenteen määrittämiseksi. |
| HTTP | Verkkoliikenneprotokolla. |
| JavaScript | Ohjelmointikieli logiikan lisäämiseksi muun muassa verkkosivuihin. |
| JSON | Standardisoitu JavaScript-oliomalli. |
| JWT | JSON-pohjainen menetelmä käyttöoikeustietueiden hallintaan. |
| Node.js | Taustajärjestelmän JavaScript-ajoympäristö. |
| npm | JavaScript-apukirjastojen hallintatyökalu. |
| Postman | Sovellus, jota käytetään muun muassa API:en testaamiseen. |
| PrimeFlex | Apukirjasto käyttöliittymän rakenteen hallintaan. |
| PrimeVue | Vue.js-kehyksen apukirjasto käyttöliittymäkomponenteille. |

| | |
|---|---|
| SFC | Yhden tiedoston komponentti. |
| SPA | Yhden sivun sovellus. |
| uuid | 128-bittinen uniikki tunniste. |
| Vuelidate | Syötteiden validointiin käytettävä Vue.js-apukirjasto. |
| Vuex | Vue.js-apukirjasto muuttujien tilanhallinnan keskittämiseen. |
| Vue.js / Vue | JavaScript-ohjelmistokehys käyttöliittymän toteuttamiseen. |

# 1 Johdanto

Teknologian kehittyessä yhä suurempi osa yhteiskunnan toiminnoista on siirtynyt digitaaliseen aikaan. Tällä vuosituhannella internetiin on siirtynyt muun muassa pankki-, vero- ja lupa-asiointi. Myös vaateostokset, lääkärin ajanvaraus, junalipun ostaminen ja koulutehtävät suoritetaan digitaalisesti. Jopa ruokakaupat ja ravintola-ala ovat kokeneet mullistuksen; ruoka tilataan mobiilisovelluksilla kotiovelle (Ahuja ym. 2021). Yhteiskunnan toimintojen ja osa-alueiden digitaalisilla siirtymillä on ainakin yksi yhdistävä tekijä: web-sovellus.

Sovellusten ja digitaalisten työkalujen lisääntyessä nykyihmisen ajanhallinta voi olla haastavaa. Ajanhallinnan helpottamiseksi voidaan käyttää tehtävienhallintajärjestelmää. Sen avulla voidaan keskittää moneen sovellukseen hajautetut tehtävät ja aikataulut yhteen järjestelmään, jolloin arjen suunnittelusta tulee mutkattomampaa.

Tässä opinnäytetyössä toteutetaan tehtävienhallintajärjestelmä nykyaikaisena web-sovelluksena full stack -periaatteella. Full stack -kehittäminen tarkoittaa web-sovelluksen kaikkien tasojen toteuttamista, käyttöliittymästä tietokantaan (W3Schools). Työn toimeksiantaja on Kuuki Tech, joka on ohjelmisto- ja sovelluskehitykseen erikoistunut yritys. Kuuki Tech on osa Kuuki Marketing Lab Oy:ta, joka on vuonna 2017 perustettu monikanavaisen markkinoinnin ja myynnin asiantuntijaorganisaatio. Kuuki työllistää 27 oman alansa ammattilaista Lahdessa ja Helsingissä, ja sen liikevaihto vuonna 2021 oli 2,73 miljoonaa euroa (Kuuki Marketing Lab Oy 2022).

Työn päätavoite on toteuttaa tehtävienhallintasovellus, jota voi käyttää ajasta ja paikasta riippumatta erilaisilla laitteilla. Päätavoitteen saavuttamiseksi tehtävienhallintajärjestelmän käyttäjän on voitava rekisteröityä sovellukseen. Rekisteröityneen käyttäjän on voitava kirjautua sovellukseen, ja kirjauduttaan sisään hänen on voitava luoda, lukea, muokata ja poistaa sekä projekteja että tehtäviä. Luoduilla tehtävillä tulee olla nimi, kuvaus, prioriteettitaso, tila ja tietokannasta valittu käyttäjä, joka vastaa tehtävän toteuttamisesta. Opinnäytetyön muita tavoitteita ovat käyttäjän tietojen tietoturvallinen käsittely, käyttäjäystävällisen käyttöliittymän sekä käyttäjäkokemusta parantavien lisätoimintojen toteuttaminen.

Toimeksiantaja asetti rajoitteet sovelluksen toteutuksessa käytettäville teknologioille. Tehtävienhallintajärjestelmän käyttöliittymä tulee toteuttaa Vue.js-ohjelmistokehyksellä ja sen Vuex- ja Primevue-apukirjastoilla. Sovelluksen taustajärjestelmä tulee toteuttaa Express-kehyksellä ja tietokanta MariaDB:llä.

Tässä työssä käsitellään aluksi modernin web-sovelluksen ja SPA-sovelluksen teoriaa, jonka jälkeen luodaan katsaus Vue.js- ja Express-ohjelmistokehyksiin sekä full stack -kehittämiseen. Sen jälkeen käsitellään tehtävienhallintajärjestelmän suunnittelua ja

toteutusta. Lopuksi analysoidaan työn tavoitteiden täyttymistä ja pohditaan tehtävienhallintajärjestelmän jatkokehitystä.

# 2 Moderni web-sovellus

## 2.1 Web-sovellukset

Ennen internetin yleistymistä tyypillinen tietokoneohjelmisto toimi siinä tietokonejärjestelmässä, mihin se oli asennettu, ilman yhteyttä internetiin. Ohjelmiston käyttämiseen tarvittavat tiedostot asennettiin koneen kovalevylle disketeiltä tai Cd-levyiltä, jonka jälkeen ohjelmisto oli käytettävissä paikallisesti (Rice 2000).

Internetin käytön yleistyessä ihmisten välinen viestintä muuttui – sähköpostit ja tekstiviestit korvasivat paperipostin ja faksikoneen, viesti- ja videochattisovellukset korvasivat puhelut ja lähikokoukset. Jatkuvasta internetyhteydestä tuli niin arkipäiväinen ilmiö, että nykyihminen on tottunut hoitamaan työelämän ja vapaa-ajan askareet digitaalisesti mistä ja milloin vain. Web-sovellukset mahdollistivat tämän muutoksen.

Web-sovellus on tietokoneohjelmisto, joka tarvitsee toimiakseen internetyhteyden ja verkkoselaimen (Techopedia 2017). Verrattuna vanhanaikaiseen, internetyhteydettömään ohjelmistoon, web-sovelluksen toimintaperiaate on erilainen. Vanhanaikaiset ohjelmistot noudattivat yhden tason mallia, jossa käyttöliittymän esitys, sovelluslogiikka ja tiedon tallennus suoritettiin samassa tietokonejärjestelmässä.

Moderneissa web-sovelluksissa käytetään yleisesti mallia, jossa käyttöliittymä, sovelluslogiikka ja tiedon varastoiminen jaetaan eri tasoihin. Verkkoselain toimii web-sovelluksen ensimmäisenä tasona, käyttöliittymänä. Web-sovelluksen toinen taso on web-palvelin, joka toimii taustajärjestelmänä vastaten osasta sovelluslogiikan laskennallisista toimenpiteistä. Web-sovelluksen kolmas taso on tietokantapalvelin, jolle tiedot varastoidaan. (IBM Cloud Education 2020.)

Koska verkkoselain toimii web-sovelluksen käyttöliittymänä, kehitetään web-sovelluksia samoilla kielillä kuin verkkosivuja. HTML:llä (HyperText Markup Language) rakennetaan käyttöliittymän runko, jonka visuaalista ilmettä muokataan CSS:llä (Cascading Style Sheets). Ohjelmallisuuden ja logiikan lisäämiseksi käytetään JavaScript-ohjelmointikieltä.

## 2.2 SPA

Vanhempien verkkosivujen toimintaperiaate on se, että jokainen verkkosivun alasivu on oma kokonaisuutensa, jonka sisältö on ladattava palvelimelta aina, kun käyttäjä siirtyy sivulta toiselle. Vanhimmat web-sovellukset noudattavat samaa periaatetta käyttäjän siirtyessä käyttöliittymänäkymästä toiseen. Tästä seuraa, että verkkosivut ja web-sovellukset

ovat hitaita ja kuormittivat tietoverkkoja jatkuvilla HTTP-pyynnöillä (HyperText Transfer Protocol). Tämän ongelman ratkaisemiseksi kehitettiin SPA-sovellukset.

SPA on lyhenne englanninkielisistä sanoista Single Page Application, ja se tarkoittaa yhden sivun sovellusta. Kun käyttäjä avaa SPA-sovelluksen verkkoselaimessa, sovellus lataa verrattain suuren määrän resursseja palvelimelta. Kun käyttäjä liikkuu SPA-sovelluksessa käyttöliittymänäkymästä toiseen, HTML-, CSS- ja JavaScript-koodi on tallessa selaimessa, ja näkymän vaihto tapahtuu ilman tarvetta ladata koko sivua uudelleen.

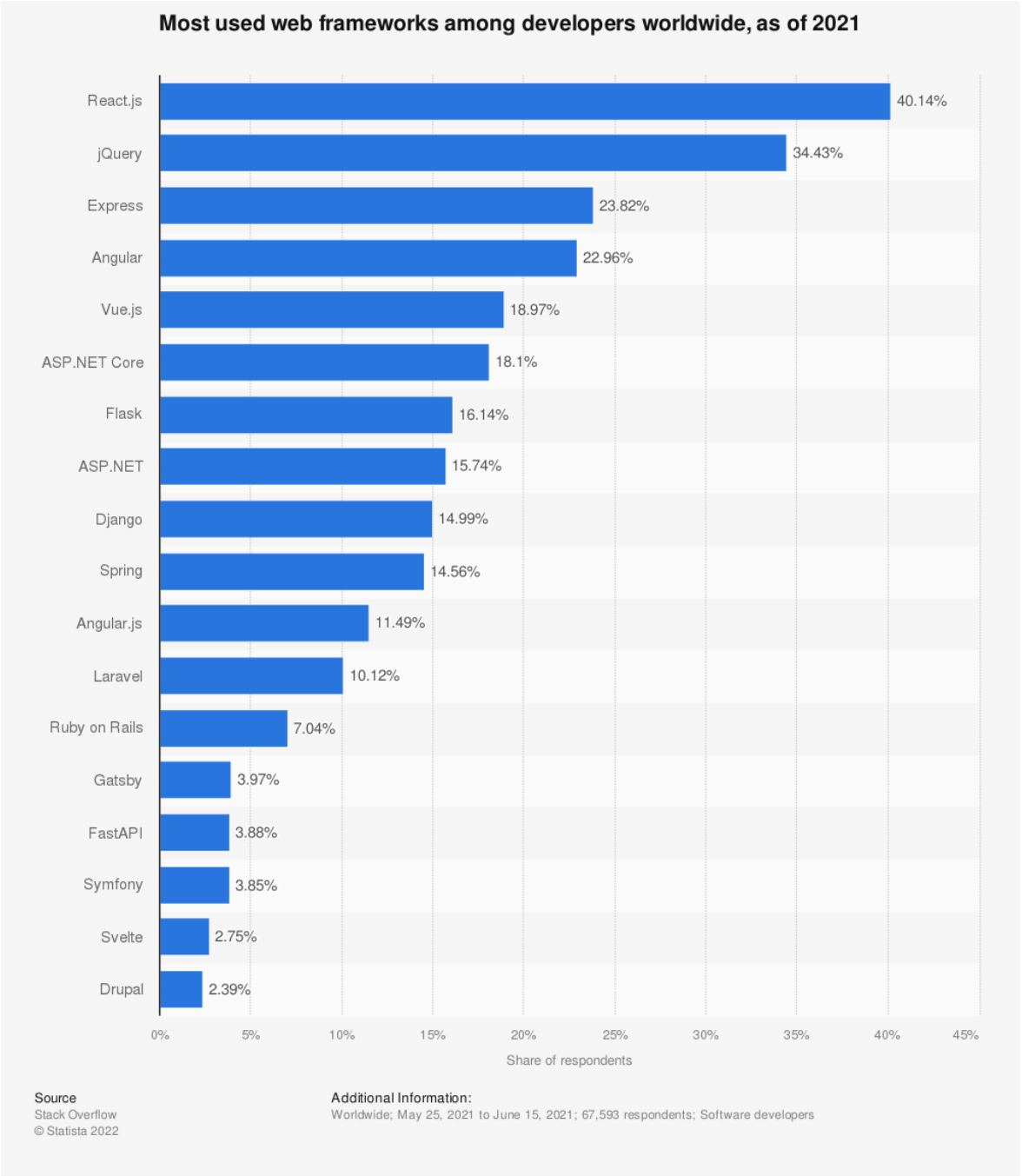
SPA-sovelluksille on ominaista, että sovellusta avatessa kestää hetken, ennen kuin tarvittavat tiedot on ladattu palvelimelta. Tämän jälkeen latausajat ovat erittäin lyhyitä tai olemattomia. Yhteys taustajärjestelmään luodaan ja pyyntö palvelimelle lähetetään vain silloin, kun sille on tarvetta. Näin vältetään turhat palvelinpyynnöt ja tietokantayhteydet ja saadaan sovelluksen toimintoihin lyhyemmät vasteajat.

SPA-sovellusten kehittämistä voidaan nopeuttaa käyttämällä ohjelmistokehyksiä. SPA-sovellusten käyttöliittymä voidaan rakentaa esimerkiksi React.js-, Angular- tai Vue.js-kehyksellä. Yhden sivun sovelluksen taustajärjestelmä voidaan toteuttaa muun muassa Node.js-ajoympäristöllä ja Express-ohjelmistokehyksellä.

## 2.3 Vue.js

Vue.js on avoimen lähdekoodin JavaScript-ohjelmistokehys, jota käytetään muun muassa SPA-verkkosivujen ja web-sovellusten käyttöliittymien toteuttamiseen. Vue-kehyksen luoja, Evan You, oli aiemmin käyttänyt Googlen AngularJS-kehystä, mutta piti sitä liian monimutkaisena pienempien sovellusten ja sovellusprototyyppien toteuttamiseen. Hän halusi luoda kevyemmän ohjelmistokehyksen, jossa yhdistyisi AngularJS.n parhaat puolet, kuten DOM:n (Document Object Model, suom. dokumenttioliomalli) manuaalisen muokkaamisen välttäminen tietojen sidonnalla (engl. data binding). (Cromwell 2016.)

Vuen ensimmäinen versio julkaistiin vuonna 2013, ja sen suosio kevyenä käyttöliittymäkehyksenä on kasvanut vuosien saatossa. Stack Overflow -sivuston vuonna 2021 tekemän kyselyn mukaan Vue on nykyään yksi suosituimmista web-ohjelmistokehyksistä. Kyselyssä ohjelmistokehittäjiltä kysyttiin mitä kehyksiä ja apukirjastoja he ovat käyttäneet kattavasti viimeisen vuoden aikana, ja mitä kehyksiä he haluavat käyttää seuraavan vuoden aikana. Kyselyyn vastasi 67593 ohjelmistokehittäjää, ja sen tulokset ovat nähtävissä kuvassa 1. (Stack Overflow 2021.)




Kuva 1. Suosituimmat web-ohjelmistokehykset (Statista 2022)

### 2.3.1 Käyttöönotto

Vue otetaan käyttöön asentamalla kehyksen rakentamistyökalu sovelluskehittämiseen käytettävälle tietokoneelle. Vue-kehyksen voi myös ottaa käyttöön sisällyttämällä kehyksen URL-osoite HTML-tiedoston script-elementtiin, mutta tässä tapauksessa sovellusprojektissa ei voida käyttää kehyksen toiminnalle keskeistä SFC-mallia. SFC-mallia käsitellään

tarkemmin seuraavassa alaluvussa Toimintaperiaate. Esimerkki Vue-kehyksen käyttöönotosta ilman rakentamistyökalun asentamista on nähtävissä kuvassa 2.

```html
<script src="https://unpkg.com/vue@3"></script>

<div id="app">{{ message }}</div>

<script>
  Vue.createApp({
    data() {
      return {
        message: 'Hello Vue!'
      }
    }
  }).mount('#app')
</script>
```

Kuva 2. Vue.js-kehyksen käyttöönotto ilman rakentamistyökalua (Vuejs a)

Vue-kehyksen käyttöönottoon rakentamistyökalulla tarvitaan Node.js-ajoympäristö ja npm-paketinhallintatyökalu. Npm on lyhenne sanoista Node Package Manager. Npm oli alun perin Node.js-ajoympäristön apukirjastojen hallintaan tarkoitettu työkalu. Sittemmin siitä on tullut yleinen JavaScript-apukirjastojen hallintatyökalu, jolla voi asentaa sovellusprojektiin kirjastoja esimerkiksi käyttöliittymän rakentamiseen ja hallintaan. Kun Node.js ja npm on asennettu, Vue-sovellusprojektin luominen aloitetaan komentorivikomennolla, joka näkyy kuvassa 3.

```sh
> npm init vue@latest
```

Kuva 3. Komento Vue-projektin luomiseksi (Vuejs b)

Komento asentaa ja käynnistää create-vuen, joka on Vue-projektien initialisointityökalu. Komennon käynnistämä create-vue avaa komentoriviin näkymän, jossa nimetään projekti ja valitaan osa käytettävistä apukirjastoista. Create-vuen valintanäkymä on nähtävissä kuvassa 4.

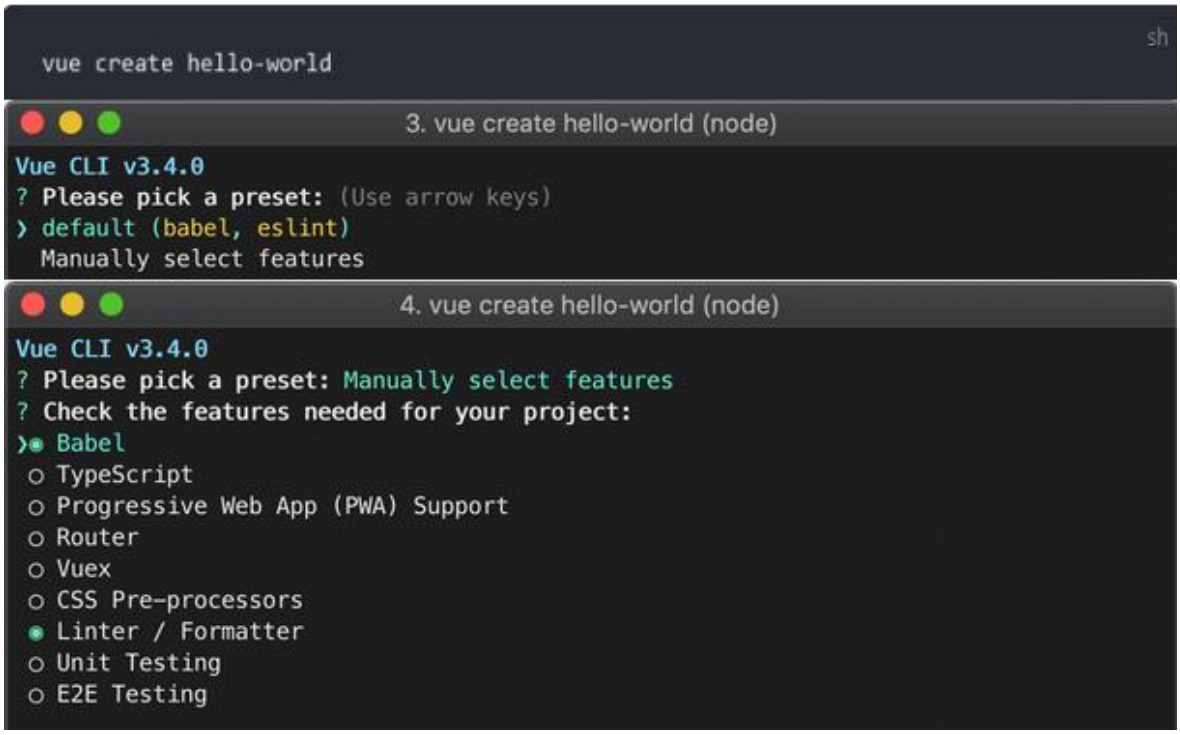
```sh
✓ Project name: … <your-project-name>
✓ Add TypeScript? … No / Yes
✓ Add JSX Support? … No / Yes
✓ Add Vue Router for Single Page Application development? … No / Yes
✓ Add Pinia for state management? … No / Yes
✓ Add Vitest for Unit testing? … No / Yes
✓ Add Cypress for both Unit and End-to-End testing? … No / Yes
✓ Add ESLint for code quality? … No / Yes
✓ Add Prettier for code formatting? … No / Yes

Scaffolding project in ./<your-project-name>...
Done.
```

Kuva 4. Create-vuen valintanäkymä (Vuejs b)

Vue-projektin voi myös alustaa Vue CLI:lla (Vue Command Line Interface). Vue CLI on Vuen oma komentorivityökalu, jossa on sovellusprojektin luomiseen tarkoitettu create-vuen kaltainen valikkonäkymä. Kun kehittäjä luo Vue-projektin Vue CLI:n avulla, hän voi valita joko oletusmuotoiset riippuvuudet tai valita ne manuaalisesti. Komentorivikomento, jolla Vue CLI asennetaan, näytetään kuvassa 5. Kuvassa 6 näytetään komento, jolla Vue CLI:ssä luodaan projekti, ja valikkonäkymät projektin riippuvuuksien hallintaan.

```sh
npm install -g @vue/cli
# OR
yarn global add @vue/cli
```

Kuva 5. Komento Vue CLI:n asentamiseksi (Vue CLI a)



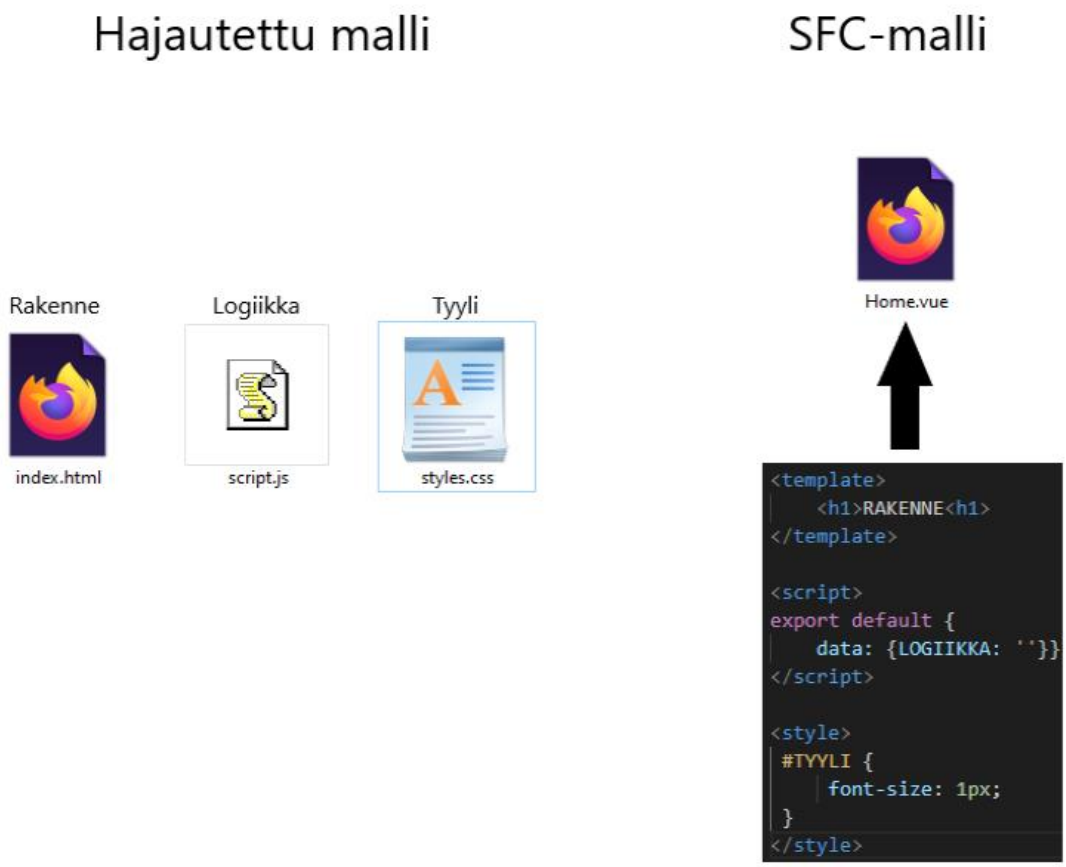

Kuva 6. Projektin luominen Vue CLI:llä (mukailtu Vue CLI b)

### 2.3.2 Toimintaperiaate

Vue-sovellus toimii SPA-sovelluksen tavoin. Kun sovellus avataan verkkoselaimessa, suuri osa sovelluksen toimintaan tarvittavista tiedostoista ladataan selaimeen. Tämän jälkeen liikkuminen näkymästä toiseen tapahtuu ilman merkittäviä latausaikoja. Yhteys web-palvelimeen ja tietokantaan luodaan vain, kun sovellus tarvitsee taustajärjestelmän toiminnallisuutta tai tietoja tietokannasta.

Vue-sovelluksen toimintaperiaate perustuu SFC-mallille. SFC on lyhenne englanninkielisistä sanoista Single File Component, joilla tarkoitetaan yhden tiedoston komponenttia. Komponentilla tarkoitetaan sovelluksen käyttöliittymän osaa, esimerkiksi navigaatiovalikkoa tai sisäänkirjautumisnäkymää.

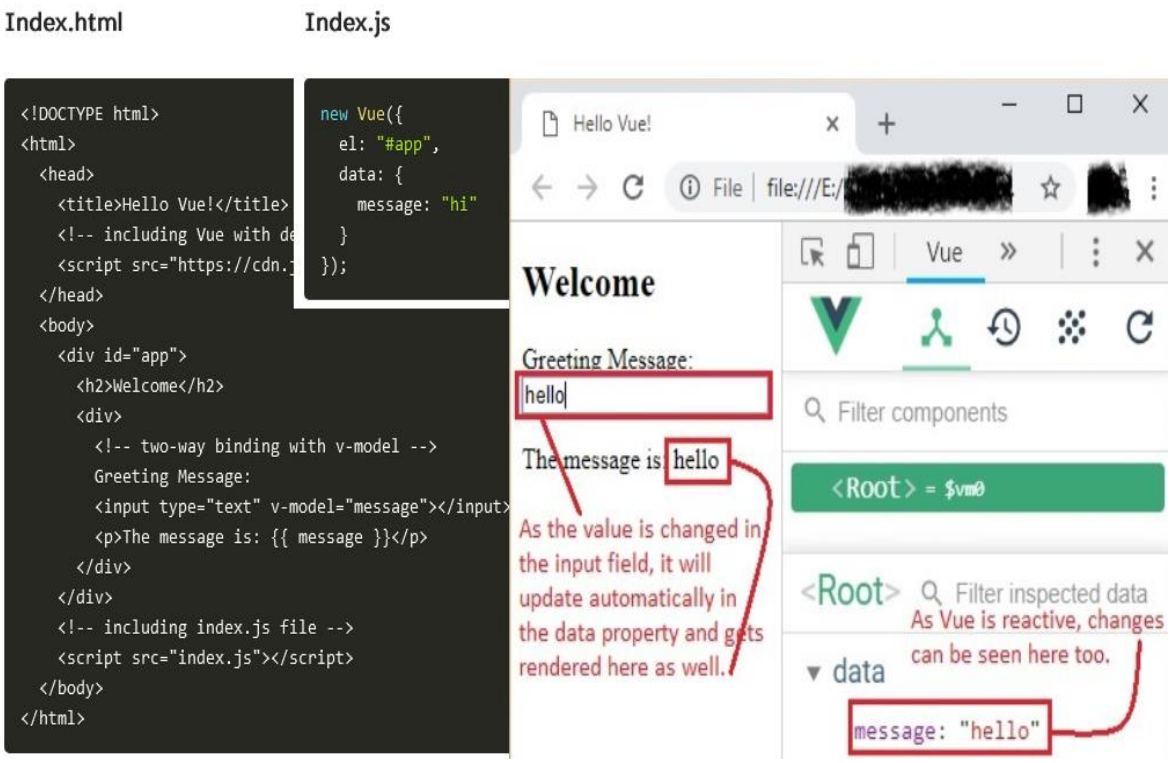
Yhden tiedoston komponentissa komponentin rakenne (HTML), tyyli (CSS) ja logiikka (JavaScript) säilytetään yhdessä tiedostossa. Ilman SFC-mallia jokaisen komponentin rakenne, tyyli ja logiikka hajautetaan erillisiin tiedostoihin. SFC:n ja hajautetun mallin väliset erot on havainnollistettu kuvassa 7.




Kuva 7. Hajautettu malli ja SFC-malli

Kun käyttöliittymäkomponentin rakenne, tyyli ja logiikka on keskitetty samaan tiedostoon, eikä komponenttia muuttaessa tarvitse etsiä sen osia eri tiedostoista, sovelluksen kehittäminen voi tehostua. SFC-mallin käyttö voi kuitenkin olla haastavaa, jos useampi kehittäjä työstää samanaikaisesti samaa komponenttia. Vue-kehys ei pakota kehittäjää valitsemaan hajautetun tai SFC-mallin väliltä, vaan sama sovellusprojekti voi sisältää sekä SFC-komponentteja että hajautetun mallin mukaisia tiedostoja. Se, mihin malliin sovelluskehityksessä päädytään, riippuu sekä projektin luonteesta että kehitystiimin työnjaosta.

Tiedon sitominen on yksi Vue-sovelluksen keskeisistä piirteistä. Vue antaa sovelluskehittäjän lisätä HTML-koodiin kehyksen omia direktiivejä. Direktiiveillä liitetään ohjelmallisten muuttujien tiloja HTML-elementteihin, jolloin elementit muuttuvat dynaamisesti muuttujien tilojen muuttuessa. Kun muuttujan tilassa tapahtuu muutos, Vue-kehys päivittää näkymän. Esimerkki direktiivien käyttämisestä sovellusnäkymän päivittämiseen on nähtävissä kuvassa 8.




Kuva 8. Esimerkki v-model-direktiivin käytöstä (mukailtu Chaitanya 2020)

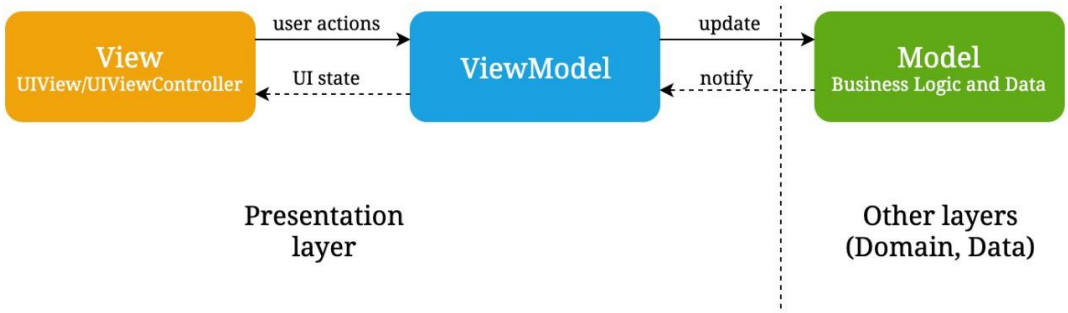
Kuvan 8 esimerkissä Index.js-tiedostossa on alustettu message-muuttujan arvoksi merkkijono ”hi”. Index.html-tiedostossa tekstisyötekenttä on sidottu message-muuttujaan v-model-direktiivillä. Kun HTML-sivu avataan verkkoselaimessa, ja syötetään merkkijono ”hello” syötekenttään, message-muuttujan arvo muuttuu tietojen sidonnan takia ”hello”:ksi. Näkymä päivittyy, ja syötekentän alapuolelle sijoitettu tervehdys päivittyy käyttäjän syötteen mukaiseksi.

Yleisesti käytettyjä Vue-direktiivejä ovat:

- v-for, jota käytetään listan tai matriisin läpikäyntiin
- v-if, jota käytetään HTML-elementin näyttämiseen tietyn ehdon täyttyessä
- v-model, jota käytetään käyttäjän syötteen yhdistämiseksi muuttujan tilaan
- v-bind, jota käytetään muuttujan tilan yhdistämiseksi HTML-elementtiin.

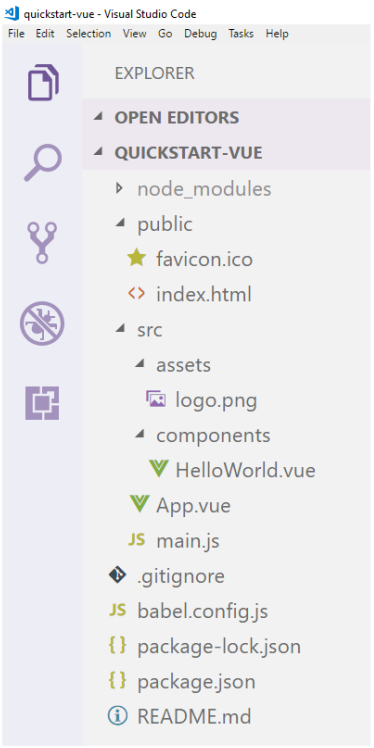
### 2.3.3 Arkkitehtuuri

Vue-sovelluksen arkkitehtuuri noudattaa MVVM-mallia (Model-View-ViewModel). Vue-kehyksen MVVM-mallin toimintaperiaate on havainnollistettu kuvassa 9.




Kuva 9. MVVM-mallin toimintaperiaate (Shcheglov 2019)

Vue-sovelluksen rakenne noudattaa SPA-periaatetta. Sovelluksella on yksi HTML-tiedosto, jonka koodiin on upotettu yksi elementti koko sovellusta varten. Tämä tiedosto on nimeltään ”index.html” ja se löytyy projektin public-kansiosta. Kuvassa 10 näkyy Vue-sovelluksen oletusmuotoinen kansiorakenne.




Kuva 10. Vue-sovelluksen oletusmuotoinen kansiorakenne (Imsirovic 2018)

Projektin src-kansiosta löytyy main.js-tiedosto, joka lataa kaikki tarvittavat kehykset, apukirjastot ja niiden käyttöliittymäkomponentit node_modules-kansiosta. Tässä tiedostossa

luodaan Vue-instanssi, rekisteröidään ladatut riippuvuudet sovelluksen käyttöön, ja lopuksi syötetään luodun sovelluksen tiedot index.html-tiedostoon div-elementtiin. Index.html-tiedoston sisältö näytetään kuvassa 11.

public > index.html > ...

```
1   <!DOCTYPE html>
2   <html lang="">
3     <head>
4       <meta charset="utf-8">
5       <meta http-equiv="X-UA-Compatible" content="IE=edge">
6       <meta name="viewport" content="width=device-width,initial-scale=1.0">
7       <link rel="icon" href="<%= BASE_URL %>favicon.ico">
8       <title><%= htmlWebpackPlugin.options.title %></title>
9     </head>
10    <body>
11      <div id="app"></div>
12      <!-- built files will be auto injected -->
13    </body>
14  </html>
15
```

Kuva 11. Index.html-tiedosto

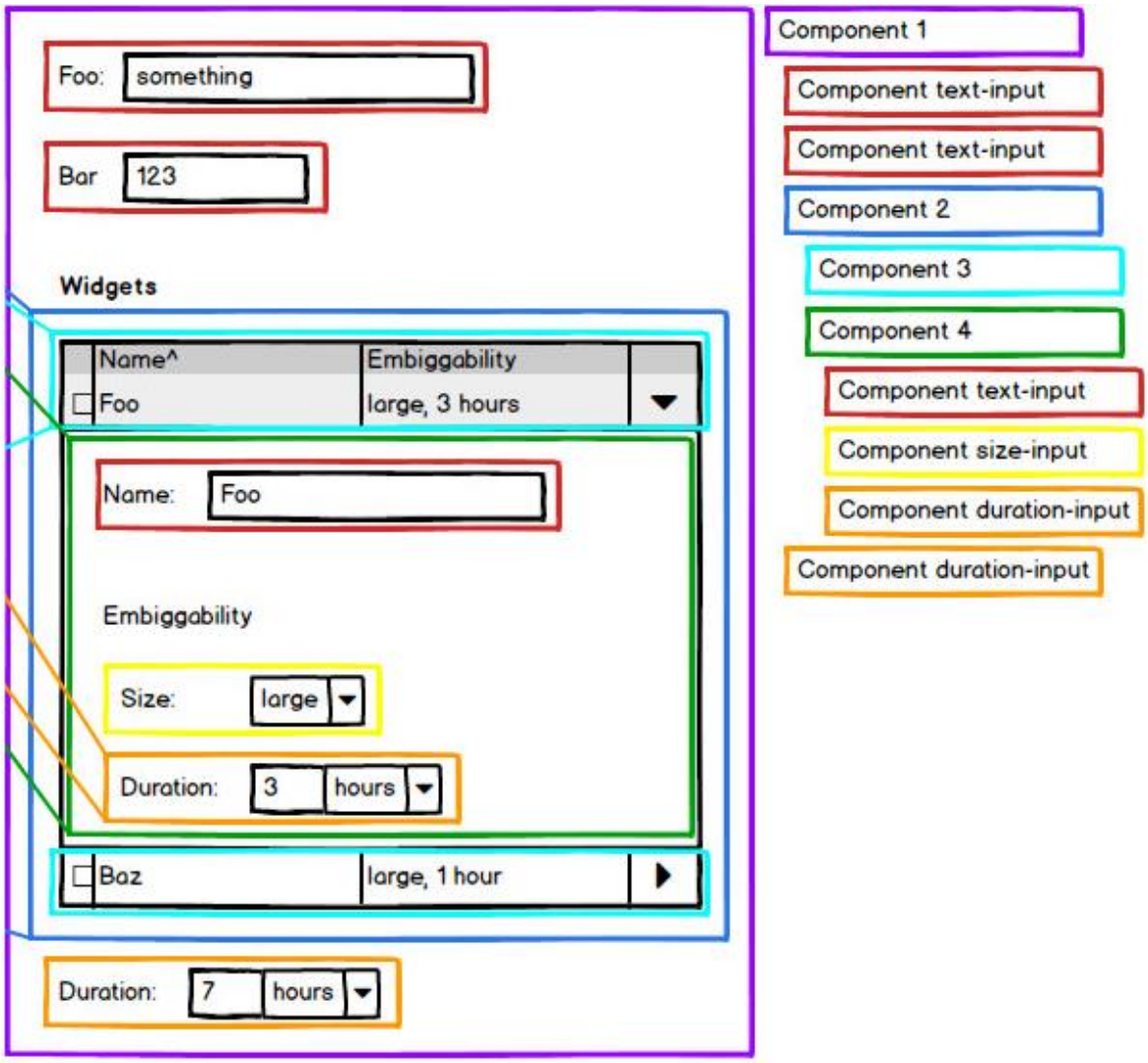
Näkymä, joka piirtyy selaimeen sen avatessa index.html-tiedoston, löytyy src-kansiosta ja on nimeltään App.vue. Sovelluksessa käytettävät näkymät upotetaan App.vue-näkymään HTML-elementteinä. Näkymät säilytetään src-kansion alakansiossa, jonka nimen sovelluskehittäjä voi valita itse. On hyvien käytänteiden mukaista nimetä alikansio mahdollisimman kuvaavasti, esimerkiksi ”screens” tai ”views”. Näkymätiedoston ensimmäisen kirjaimen tulee olla iso ja tiedostopääte on ”.vue”, esimerkiksi Home.vue.

Sovelluksen näkymiä voi upottaa toistensa sisälle. Jos edellä mainitussa Home.vue-näkymässä tulee olla kalenterikomponentti, kalenterista luodaan ”.vue”-päätteinen tiedosto, joka tuodaan Home.vue-tiedoston script-osioon ja upotetaan HTML-elementtinä sen template-osioon. Kalenterikomponentin tuominen ja upotus Home.vue-tiedostoon on nähtävissä kuvassa 12.

```
<template>
  <div>
    <h1>Hello {{ user.username }}!</h1>
    <edit-profile-form v-if="showEditProfileForm" @emitToParent3="updateParent3" />
    <div id="profileView" v-if="!showEditProfileForm">
      <div id="mycalendar">
        <my-calendar />
      </div>
      <div class="col-12 md:col-12" id="myProfile">
        <Button @click="toProfile" label="Edit Profile" class="p-button-raised p-button-help" />
      </div>
      <p></p>
    </div>
  </div>
</template>

<script>
import { mapState, mapGetters } from 'vuex';
import EditProfileForm from '../components/EditProfileForm.vue';
import MyCalendar from '../components/MyCalendar.vue';

export default {
  components: { EditProfileForm, MyCalendar },
  data() {
    return {
      username: '',
      showEditProfileForm: false
    };
  },
```

Kuva 12. Kalenterikomponentti Home.vue-tiedostossa

Sisäkkäisillä komponenteilla saadaan luotua modulaarinen sovellus, joka koostuu pienistä, uudelleen käytettävistä osista. Vue-kehys mahdollistaa komponenttien upottamisen monessa tasossa, eli lapsikomponentilla voi olla omia lapsikomponentteja. Kuvassa 13 on havainnollistettu useassa tasossa toisiinsa upotettujen komponenttien välisiä suhteita.




Kuva 13. Esimerkki sisäkkäisistä Vue-komponenteista (mukailtu AndreKR 2017)

Vue-kehyksessä muuttujien tilat lähetetään (engl. emit) tarvittaessa emokomponentilta lapsikomponentille props-oliossa. Mitä monimutkaisemmaksi sovellus kehittyy, ja mitä enemmän komponentteja on sisäkkäin, sitä vaikeampi on ylläpitää komponenttien välisiä suhteita ja muuttujien tiloja. Näiden suhteiden ja muuttujien tilojen hallintaan voidaan käyttää esimerkiksi Vuex-apukirjastoa.

Vuex-kirjaston avulla voidaan keskittää koko sovellusta koskevien muuttujien tilat yhteen tiedostoon, jota kutsutaan storeksi (suom. varasto). Vuex-store yhdistetään komponentteihin tuomalla storesta halutut muuttujatilat ja funktiot komponenttien script-osioon. Sovelluksen laajentuessa storeja voi olla useampia, esimerkiksi yksi käyttäjän tilojen hallintaan ja toinen käyttäjän kirjoittamien viestien tilojen hallintaan.

## 2.4 Express

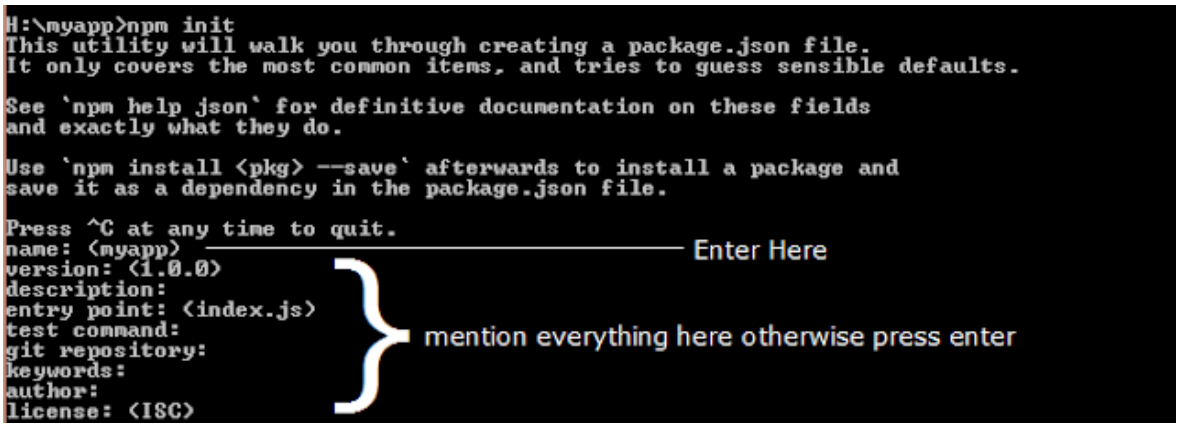
Express on Node.js-ajoympäristön avoimen lähdekoodin taustajärjestelmäkehys. Node.js pohjautuu Googlen V8 JavaScript-moottorille, jonka tehtävä on suorittaa JavaScript-koodia. Perinteisesti JavaScript-koodi suoritetaan käyttäjän verkkoselaimessa, taustajärjestelmien ja palvelimien käyttäessä muita ohjelmointikieliä, kuten Pythonia, Rubya, Javaa tai PHP:tä. Node.js-ajoympäristön avulla sekä verkkoselain että taustajärjestelmä ohjelmoidaan JavaScriptillä, mikä voi parantaa taustajärjestelmän suorituskykyä ja tehostaa sovelluksen kehittämistä (Obrizan 2020).

Express-taustajärjestelmäkehys luotiin minimalististen ja kevyiden web-sovelluspalvelimien toteuttamiseksi. Kehystä käytetään myös API:en, eli ohjelmointirajapintojen, rakentamiseen. API on lyhenne englannin kielen sanoista Application Programming Interface. Ohjelmointirajapinnat mahdollistavat eri ohjelmien välisen viestinnän ja tiedonvaihdon. API:n voi mieltää rajaksi ohjelmien tai ohjelman ja komponentin välillä.

Express-kehyksen minimalistisuus tarkoittaa sitä, että sen oletusversiossa ei ole suurta määrää toiminnallisuutta. Ajatuksena on se, että Express-kehys sisältää vain välttämättömät toiminnot, ja kehystä käyttävä ohjelmoija lisää siihen tarvitsemansa liitännäiset (engl. plugin) ja väliohjelmistot (engl. middleware). Näin ollen Express-kehyksellä toteutettu taustajärjestelmä pysyy kevyenä, ja sisältää vain ne toiminnallisuudet, joita tarvitaan.

### 2.4.1 Käyttöönotto

Kuten Vue-kehys, myös Express tarvitsee toimiakseen Node.js-ajoympäristön. Kun Node.js on asennettu, luodaan Express-sovellukselle kansio. Tähän kansioon luodaan package.json-tiedosto sovelluksen riippuvuuksia varten komennolla ”npm init”. Komentoriville ilmestyy kehotteita, joiden mukaisesti ohjelmoijan tulee muun muassa nimetä sovellus. Komentorivin kehotteet näkyvät kuvassa 14.




Kuva 14. Kehotteet ”npm init”-komennon jälkeen (Bikash)

Kohtaan ”entry point” ohjelmoijan tulee kirjoittaa sovelluksen päätiedoston nimi. Oletusarvona on tiedosto ”index.js”. Kun päätiedosto on nimetty, asennetaan Express-kehys projektikansioon komennolla ”npm install express”. Tämän komennon myötä Express asentuu sovellusprojektiin ja sen riippuvuuslistaan package.json-tiedostoon. Kun Express on asennettu, sovelluksen päätiedostoon on ilmestynyt JavaScript-koodia. Oletuskoodi on nähtävissä kuvassa 15.

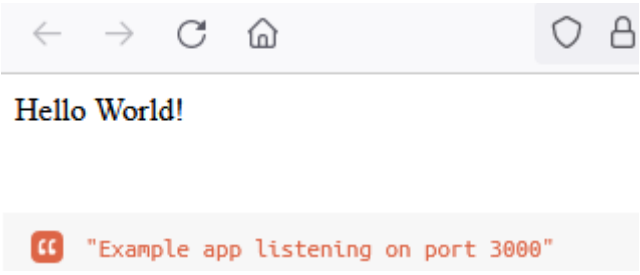
```
1  const express = require('express' 4.17.3 )
2  const app = express()
3  const port = 3000
4
5  app.get('/', (req, res) => {
6    res.send('Hello World!')
7  })
8
9  app.listen(port, () => {
10   console.log(`Example app listening on port ${port}`)
11 })
```

Kuva 15. Express-sovelluksen päätiedoston oletuskoodi (Expressjs a)

Koodin ensimmäisillä riveillä Express-kehys sijoitetaan vakioon ”express” ja Express-instanssi vakioon ”app”. Kuunneltavan portin numero sijoitetaan vakioon ”port”. Funktio ”app.get()” määrittää miten palvelin käsittelee reitistä ”/” tulevat HTTP-GET-pyynnöt. Tässä tapauksessa GET-pyyntöön vastataan lähettämällä takaisin merkkijono ”Hello World!”.

Lopuksi Express-sovellus määrätään kuuntelemaan porttia 3000 ja kirjaaman ilmoituksen konsolilokiin.

Express-palvelinsovellus käynnistetään komennolla ”node sovelluksen-nimi.js”. Kuvassa 16 näytetään käynnistetyn Express-palvelinsovelluksen konsoliloki ja verkkoselaimen näkymä osoitteessa http://localhost:3000/.



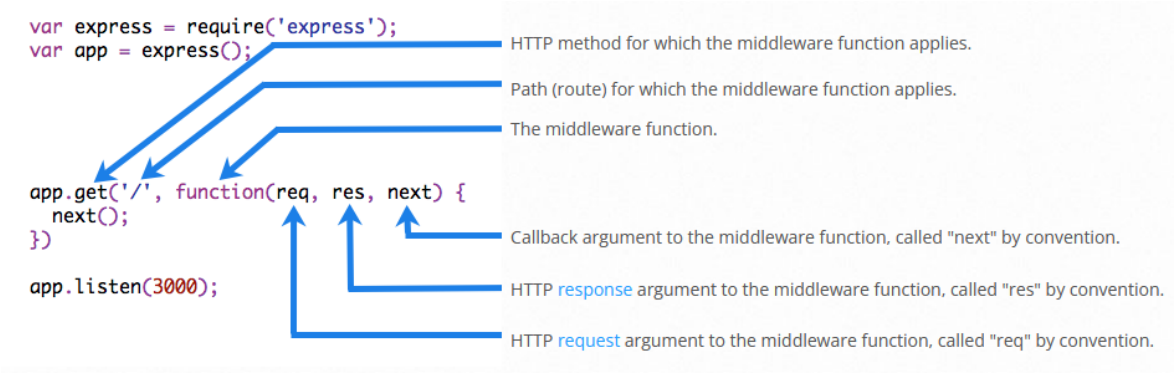

Kuva 16. Sovelluksen selainnäkymä ja konsolilokiviesti (mukailtu Expressjs a)

Esimerkkinä käytetty web-palvelin edustaa Express-sovellusta kevyimmillään. Monipuolisemman sovellusrungon luomiseksi voidaan käyttää esimerkiksi express-generator-työkalua. Työkalu ladataan komentorivikomennolla ”npm install -g express generator” ja käynnistetään komennolla ”express”.

### 2.4.2 Toimintaperiaate

Express-ohjelmistokehyksen toimintaperiaatteen keskiössä ovat reitit, HTTP-metodit ja väliohjelmistot. Yleisiä HTTP-metodeja ovat POST, GET, PUT ja DELETE. POST-metodilla lähetetään tietoa selaimesta taustajärjestelmään, ja GET-metodilla pyydetään tietoa taustajärjestelmästä. PUT-metodia käytetään taustajärjestelmästä löytyvän tiedon korvaamiseen ja DELETE-metodia taustajärjestelmästä löytyvän tiedon poistamiseen.

Express-sovelluksessa määritellään, miten eri reiteistä tulevia HTTP-pyyntöjä käsitellään. Reitti voi olla esimerkiksi verkkosivu tai REST API-päätepiste. Erilaisten HTTP-metodien reittikohtainen käsittely määritellään Express-kehyksen funktiolla ”app.METODI(REITTI, FUNKTIO)”. Funktion rakenne on esitetty kuvassa 17.




Kuva 17. Funktio Express-sovelluksessa (mukailtu Expressjs b)

Express-sovelluksen ”(app.METODI)”-funktiolla on callback-funktio. Callback-funktiolla tarkoitetaan funktiota, joka syötetään parametrinä toiselle funktiolle. Kun ”(app.METODI)”-funktiota vastaava pyyntö saapuu palvelimelle, kutsutaan sen callback-funktiota. Callback-funktiolla on parametrit req, res ja next.

Req-olio sisältää vastaanotetun HTTP-pyynnön otsikon ja viestin. Otsikko voi sisältää muun muassa tietoa pyynnön lähettäjästä, pyynnön ajankohdan ja tietoa hyväksytyistä vastausmuodoista. Viesti voi olla esimerkiksi merkkijono, numero, lista tai olio. Tilanteen mukaan HTTP-pyynnön otsikko tai viesti voi olla tyhjä.

Res-olio sisältää Express-web-palvelimen vastauksen HTTP-pyyntöön. Nykyaikaisten verkkosivujen ja -sovellusten yleisiä palvelinpuolen vastausmuotoja ovat HTML-, XML- ja JSON-tiedostot.

Express-kehyksessä sekä req- että res-olioilla on monta sisäänrakennettua ominaisuutta ja metodia. Niitä kutsutaan kirjoittamalla koodiin ”req/res.ominaisuus/metodi()”. Esimerkiksi ominaisuus ”req.body” vastaa HTTP-pyyntöolion viestiosaa, ja metodi ”res.json({ ”user”: ”kalle” })” lähettää sulkujen sisällä olevan tiedon JSON-muotoisena vastauksena HTTP-pyyntöön. Joidenkin ominaisuuksien ja metodien käyttämiseksi vaaditaan, että oikeaa väliohjelmistoa on kutsuttu.

Minimalistisena taustajärjestelmänä Express nojaa väliohjelmistoihin toiminnallisuuden lisäämiseksi. Väliohjelmistot ovat funktioita, jotka voivat

- suorittaa koodia
- muokata req- ja res-olioita
- kutsua väliohjelmistoketjun seuraavaa funktiota
- lopettaa req-res-syklin.

Väliohjelmistoja ketjutetaan siten, että ensimmäisen funktion tehtyä tehtävänsä, se kutsuu ketjun seuraavaa funktiota. Funktion sisällä seuraavaa väliohjelmistoa edustaa muuttuja ”next”. Mikäli väliohjelmisto ei ole ketjun viimeinen funktio, se kutsuu seuraavaa väliohjelmistoa metodilla ”next()”. Muuttujan ”next” ja metodin ”next()” käyttöä väliohjelmistossa havainnollistettiin kuvassa 17.

Express-sovellukset ovat pohjimmiltaan kevyitä web-palvelinsovelluksia, jotka koostuvat väliohjelmistoketjuista. Ne voidaan jakaa viiteen ryhmään:

- Sovellustason väliohjelmistot
- Reititintason väliohjelmistot
- Virheenkäsittelyväliohjelmistot
- Sisäänrakennetut väliohjelmistot
- Kolmannen osapuolen väliohjelmistot.

Kolmannen osapuolen väliohjelmistoista on huomioitava, että ne pitää asentaa sovellukseen ennen kuin niitä voi käyttää. Asentaminen aloitetaan npm-paketinhallintatyökalun komentorivikomennolla ”npm install väliohjelmiston-nimi”. Kun npm on lisännyt riippuvuuden sovellusprojektiin, väliohjelmisto tulee asentaa ja ladata sovellukseen ennen palvelimen käynnistystä. Asennus ja lataaminen on nähtävissä kuvassa 18.

```
const express = require('express')
const app = express()
const cookieParser = require('cookie-parser')

// load the cookie-parsing middleware
app.use(cookieParser())
```

Kuva 18. Väliohjelmiston asennus ja lataaminen (Expressjs c)

## 2.5 Ohjelmistokehykset ja full stack -kehittäminen

Vuen ja Expressin kaltaiset ohjelmistokehykset ovat tehneet web-sovellusten toteuttamisesta tehokkaampaa ja nopeampaa, ja niiden suosio sovellusten toteuttamisen välineenä on kasvanut. Ohjelmistokehysten suosion kasvaessa myös niiden apukirjastojen ja väliohjelmistojen määrä on kasvanut. Näin ollen kehyksiin liittyvät ohjelmointitilanteet, joihin ei löydy valmista ratkaisua apukirjastoista, ovat vähentyneet. Valmiilla ratkaisuilla ja kehysten parhaiden käytänteiden noudattamisella voidaan kirjoittaa virheettömämpää koodia.

Jos web-sovelluskehittäjä käyttää Vue-kehystä sovelluksen käyttöliittymän toteuttamiseen, ja haluaa sijoittaa valikoita ja painikkeita sovellusnäkymään, hän voi käyttää esimerkiksi PrimeVue-apukirjastoa. PrimeVue-kirjasto sisältää valmiita käyttöliittymäkomponentteja. Tämän apukirjaston avulla sovelluskehittäjä voi nopeuttaa kehitystä, kun hänen ei tarvitse luoda jokaista käyttöliittymäkomponenttia tyhjästä. Jos sovelluksessa on syötekenttä käyttäjän sähköpostiosoitteen ja puhelinnumeron syöttämiseksi, kehittäjä voi lisätä sovellukseen Vuelidate-kirjaston syötteiden oikeellisuuden tarkastamiseksi.

Myös taustajärjestelmän toteuttaminen on nopeutunut ohjelmistokehysten tarjonnan parantuessa. Varsinkin Node.js-pohjaiset kehykset, kuten Express, tehostavat web-sovellusten toteuttamista, kun sekä käyttöliittymä että taustajärjestelmä ohjelmoidaan samalla kielellä. Kehittäjien ei tarvitse opetella uusia kieliä palvelimen toteuttamiseksi, ja sovellusten tasojen väliset yhteensopivuusongelmat vähenevät yhteisen ohjelmointikielen myötä.

Lähes kaikkia tilanteita ja sovelluksen toiminnallisuuksia varten on olemassa valmiita ratkaisuja. Siksi moderneja web-sovelluksia voidaan kehittää nopeammin ja pienemmillä resursseilla. Sovelluskehitystiimien pienentyessä yksittäiseltä sovelluskehittäjältä vaaditaan laajempaa osaamista.

Ennen saattoi riittää, että kehittäjä osasi tuottaa joko käyttöliittymän, taustajärjestelmän tai tietokannan. Nykyään monet ohjelmistokehitysyritykset noudattavat sovelluskehityksessä full stack -periaatetta. Full stack -termi kattaa sovelluksen kaikki tasot; käyttöliittymän, taustajärjestelmän ja tietokannan. Full stack -periaatteen mukaisessa ohjelmistokehityksessä yksittäiset kehittäjät toteuttavat web-sovelluksen kaikkia tasoja. Full stack -kehityksessä kehittäjiltä vaaditaan ymmärrystä sovelluksesta kokonaisuutena ja siitä, miten sovelluksen eri tasot ovat yhteydessä toisiinsa.

# 3 Tehtävienhallintajärjestelmän suunnittelu

## 3.1 Vaatimusmäärittely

Tehtävienhallintajärjestelmän toimeksiantaja määritteli, että sovelluksen tulisi olla nykyaikainen SPA-periaatetta noudattava web-sovellus, jota voi käyttää ajasta ja paikasta riippumatta. Toimeksiantaja määritteli vaatimukset järjestelmän toiminnallisuuksista seuraavasti:

- Sovelluksessa tulee olla uuden käyttäjän rekisteröinti, jossa käyttäjä syöttää sähköpostiosoitteensa ja valitsee itselleen salasanan.
- Sähköpostiosoitteet ja salasanat tulee tallentaa tietokantaan.
- Käyttäjien tunnisteet ja salasanat tulee käsitellä tietoturvallisesti.
- Sovelluksessa tulee olla rekisteröityneen käyttäjän sisäänkirjautuminen.
- Käyttäjän tulee voida luoda, lukea, muokata ja poistaa projekteja, jotka toimivat säiliöinä tehtäville.
- Projektilla tulee olla nimi, alkamispäivämäärä, loppupäivämäärä ja tila.
- Projektilla tulee olla kolme mahdollista tilaa: ”Created” (luotu), ”Started” (aloitettu) ja ”Finished” (valmis).
- Käyttäjän tulee voida luoda, lukea, muokata ja poistaa projekteihin kuuluvia tehtäviä.
- Tehtävällä tulee olla nimi, kuvaus, prioriteettitaso, vaihe, aikaraja ja vastuuhenkilö. Vastuuhenkilö on käyttäjä, joka vastaa valitun tehtävän suorittamisesta.
- Tehtävällä tulee olla kolme prioriteettitasoa: ”Low” (matala), ”Medium” (keskitaso) ja ”High” (korkea).
- Tehtävällä tulee olla neljä vaihetta: ”Backlog” (jonossa), ”Working” (työn alla), ”Testing” (testivaiheessa) ja ”Approved” (hyväksytty).

Toimeksiantaja määräsi rajoitteet sovelluksen toteuttamiseksi käytettäville teknologioille. Tarkemmat vaatimukset sovelluksessa käytettävistä teknologioista ja sen rakenteesta olivat seuraavanlaiset:

- Sovelluksen käyttöliittymä tulee toteuttaa Vue-ohjelmistokehyksellä.
- Valmiiden käyttöliittymäkomponenttien lisäämiseksi tulee käyttää PrimeVue-apukirjastoa.
- Käyttöliittymän koodirakenteen ja muuttujien tilanhallinnan tulee noudattaa Vuex-apukirjaston mallia.
- Sovelluksen taustajärjestelmä tulee toteuttaa Node.js-ajoympäristöllä ja Express-kehyksellä.
- Sovelluksen taustajärjestelmän tulee yhdistää käyttöliittymä relaatiotietokantaan.

- Relaatiotietokanta pitää toteuttaa MariaDB:lla.

Valmista tehtävienhallintajärjestelmää tullaan käyttämään sovelluskehitysprojekteissa sekä projektien että yksittäisten tehtävien hallintaan. Siksi tehtävienhallintasovelluksen tulisi olla optimoitu kannettavien tietokoneiden ja pöytätietokoneiden näytöille. Sovelluksen tarkoitetun käyttöympäristön takia sen optimointi pienemmille näytöille ei ole tärkeää.

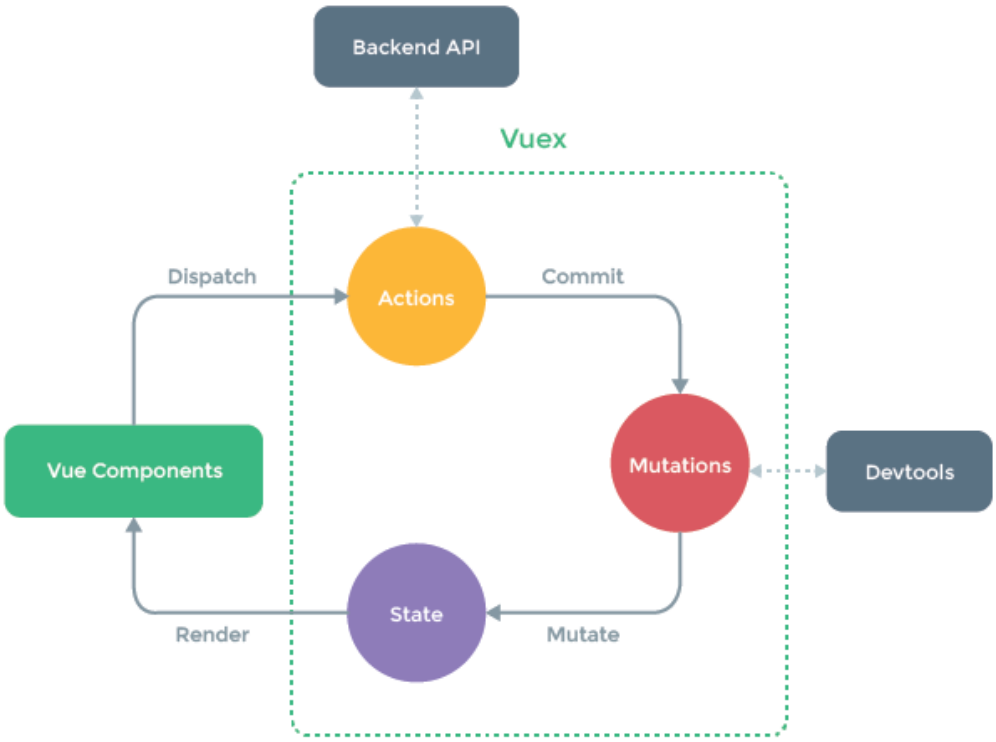
Tehtävienhallintajärjestelmän toiminnallisuus- ja teknologiavaatimusten täyttämiseksi annettiin vapaus valita apukirjastoja ja väliohjelmia tarpeiden mukaan. Toimeksiantaja antoi toimintavapauden myös käyttöliittymän suunnitteluun, kunhan sovellus on käyttäjälle selkeä ja mutkaton käyttää.

## 3.2 Käyttöliittymän suunnittelu

Käyttöliittymän suunnittelu jaettiin kahteen osaan: rakenteen suunnitteluun ja ulkoasun suunnitteluun. Molempien osien suunnittelussa pääperiaatteena oli sovelluksen toiminnallisuuksien toteuttaminen käyttäjäystävällisesti. Sekä rakenteen että ulkoasun suunnittelua ohjasivat toimeksiantajan määrittelemät rajoitteet sovelluksen toteuttamiseen käytettävistä teknologioista.

### 3.2.1 Käyttöliittymän rakenne

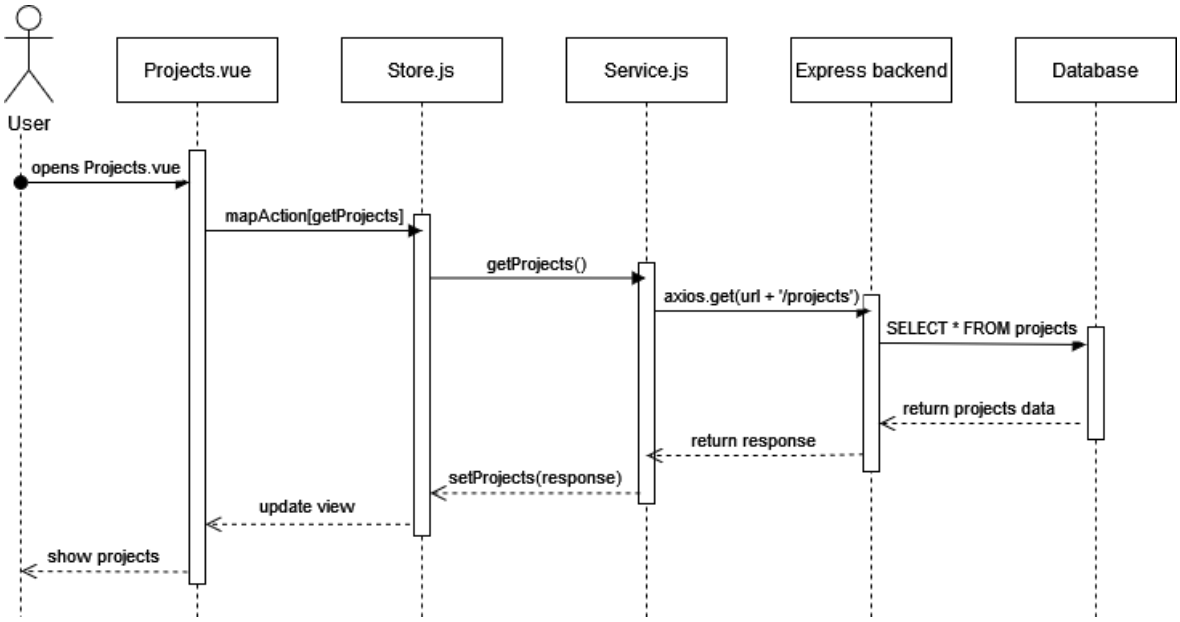
Tehtävienhallintajärjestelmän rakenne suunniteltiin noudattamaan Vuex-apukirjaston mallia, jossa ohjelmallisten muuttujien tilat keskitetään store-tiedostoon. Käyttöliittymänäkymät kutsuvat storen Getters-, Actions- ja Mutations-funktioita. Getters-funktiot palauttavat muuttujan tilan, Actions-funktiot kutsuvat muita funktioita ja Mutations-funktiot muuttavat muuttujien tiloja. Vuex-apukirjaston toimintaperiaate on havainnollistettu kuvassa 19.




Kuva 19. Vuex-apukirjaston toimintaperiaate (Vuexjs)

Jotta sovelluksen käyttöliittymän koodi ja tiedostojen rakenne pysyisivät mahdollisimman selkeinä, suunniteltiin käyttöliittymä toteutettavaksi siten, että kaikki taustajärjestelmäpyynnöt tehdään services-kansion service.js-tiedostoista.

Kun tehtävienhallintajärjestelmän käyttäjä siirtyy sovelluksen projektinäkymään, näkymä kutsuu Vuex-storen Action-funktiota, joka vuorostaan kutsuu Service.js-tiedostosta löytyvää funktiota. Service.js-tiedoston funktio lähettää pyynnön taustajärjestelmälle, joka ottaa sen vastaan ja tekee sen mukaisen tietokantakyselyn. Tietokanta palauttaa kyselyn tuloksen taustajärjestelmälle, joka lähettää tuloksen vastauksena käyttöliittymän Service.js-tiedoston funktiolle, joka lähettää sen edelleen Vuex-storeen. Vuex-storessa funktio tekee vastauksen pohjalta muutoksen projects-listan tilaan täyttämällä sen vastauksen sisältämillä projektiolioilla. Vue.js:n toimintaperiaatteiden mukaisesti käyttöliittymä havaitsee muutoksen projects-listan tilassa, ja päivittää projektinäkymän näyttämään listan käyttäjälle. Sovelluksen suunniteltu toimintaperiaate on nähtävissä kuviossa 1.

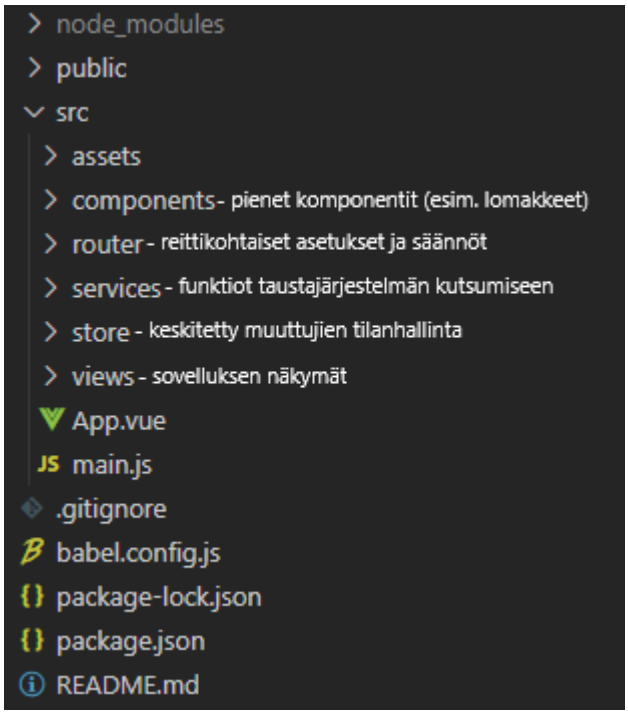



Kuvio 1. Sovelluksen suunniteltu toimintaperiaate

Tehtävienhallintasovelluksen tiedostorakenne suunniteltiin siten, että sovelluksen päänäkymät sijoitetaan views-kansioon. Pienemmät komponentit, kuten sovelluksen sisäiset lomakkeet ja valikot, sijoitetaan erilliseen components-kansioon. Erottamalla päänäkymät pienemmistä komponenteista yksittäisten tiedostojen koodimäärä pienenee ja ylläpito tehostuu. Myös pienempien komponenttien käyttäminen uudelleen eri näkymissä mahdollistuu.

Käyttöliittymän näkymien välisen liikkumisen hallitsemiseksi suunniteltiin sovelluksessa käytettävän Vue-kehyksen Vue Router-apukirjastoa. Vue Routerin avulla voidaan määritellä muun muassa URL-kohtaiset säännöt näkymille, esimerkiksi mitkä näkymät näytetään vain sisään kirjautuneille käyttäjille.

Suunnitellut kehys- ja apukirjastovalinnat muokkaavat tehtävienhallintajärjestelmän koodi- ja tiedostorakennetta. Kuvassa 20 on nähtävissä suunnitteluvaiheessa tehtyjen valintojen mukainen tiedostorakenne kansiokohtaisine selityksineen.

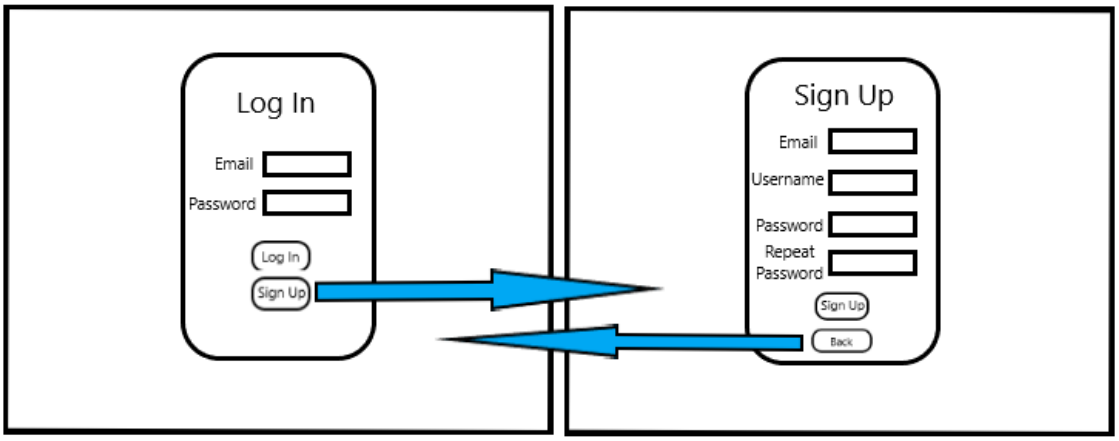



Kuva 20. Suunniteltu tiedostorakenne

### 3.2.2 Käyttöliittymän ulkoasu

Sovellusprojektin toimeksiantajan vaatimus oli, että tehtävienhallintajärjestelmän käyttöliittymä olisi selkeä ja helppokäyttöinen. Vaatimusten toteuttamiseksi käyttöliittymä suunniteltiin mahdollisimman minimalistiseksi. Kussakin näkymässä tulisi näkyä vain ne tiedot, listat ja painikkeet, joilla on näkymän hallinnan kannalta merkitystä.

Sovelluksen aloitusnäkymäksi suunniteltiin kirjautumisnäkymä. Kirjautumisnäkymästä voi kirjautua sisään sovellukseen, jos käyttäjä on rekisteröitynyt. Jos käyttäjä ei ole rekisteröitynyt, hän pääsee napin painalluksella rekisteröitymisnäkymään. Luonnos suunnitelluista kirjautumis- ja rekisteröitymisnäkymistä on esitetty kuvassa 21.

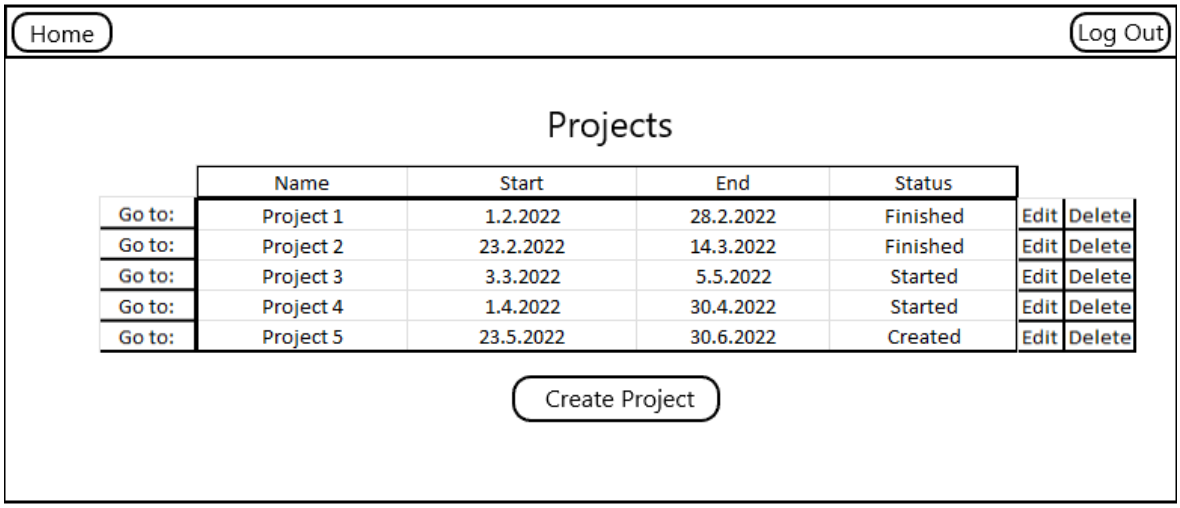



Kuva 21. Luonnos kirjautumis- ja rekisteröintinäkymistä

Kun käyttäjä on rekisteröitynyt ja kirjautunut sisään sovellukseen, hänelle avautuu henkilökohtainen kotinäkymä. Tässä näkymässä käyttäjää tervehditään tervehdystekstillä, ja hän voi siirtyä tutkimaan järjestelmään luotuja projekteja klikkaamalla Projects-painiketta. Tehtävienhallintasovelluksen toiminnallisuusvaatimukset huomioiden kotinäkymä, jossa on vain yksi painike, saattaa tuntua turhalta. Tämä näkymä on kuitenkin hyvä sisällyttää sovellukseen, koska se on looginen näkymä mahdollisten lisätoimintojen lisäämiseen.

Toisin kuin kirjautumis- ja rekisteröitymisnäkymissä, kotinäkymässä tulee olla navigaatiopalkki näkymän yläreunassa. Tästä navigaatiopalkista voi siirtyä päänäkymästä toiseen, tai kirjautua ulos sovelluksesta. Navigaatiopalkki tulee olla näkyvillä kaikissa näkymissä, joihin vaaditaan käyttäjän kirjautumista. Näin ollen uloskirjautumispainike toteutetaan vain navigaatiopalkkiin, eikä painikkeita tarvitse upottaa jokaiseen näkymään. Tämä selkeyttää sovelluksen käyttöä, kun painike löytyy näkymästä riippumatta aina samasta kohdasta.

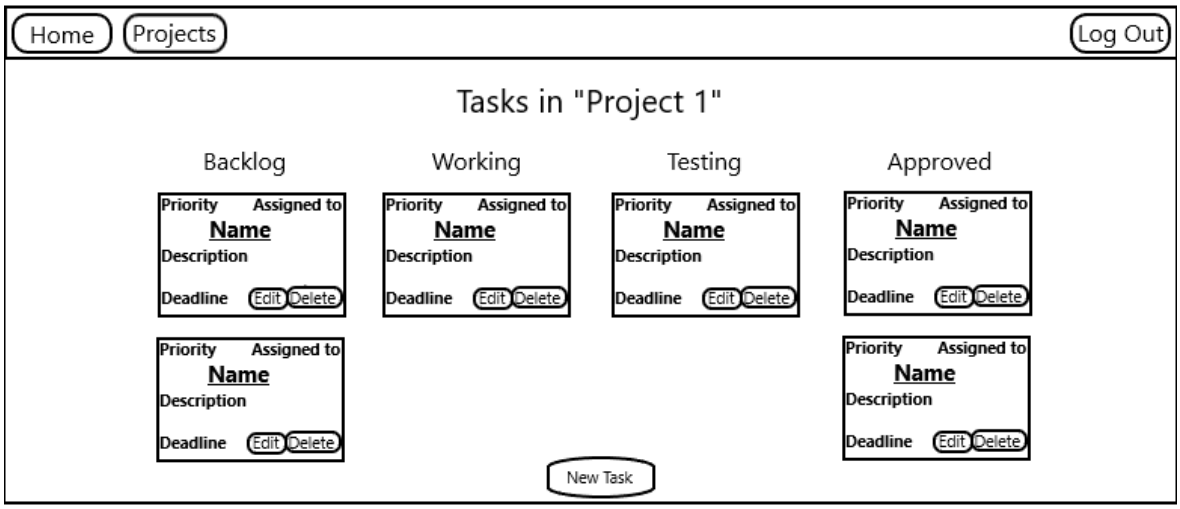
Kun käyttäjä klikkaa kotinäkymän Projects-painiketta, hän siirtyy projektinäkymään. Projektinäkymässä luodut projektit näytetään listana, josta selviää jokaisen projektin nimi, alkamispäivämäärä, loppupäivämäärä ja tila. Listalla kunkin projektin tiedot näkyvät omalla rivillään. Projektirivin vierestä tulee löytyä painikkeet projektin muokkaamiselle ja poistamiselle. Tehtävänäkymään siirrytään projektin nimen viereltä löytyvää painiketta klikkaamalla. Sivulta tulee löytyä myös painikkeet uuden projektin luomiseen ja palaamiseen takaisin kotinäkymään. Suunniteltu projektinäkymä on nähtävissä kuvassa 22.




Kuva 22. Luonnos projektinäkymästä

Klikattuaan haluamansa projektin painiketta projektinäkymässä, käyttäjä siirtyy tehtävänäkymään. Tehtävänäkymä tulee jakaa neljään alueeseen tehtävän vaiheen mukaan. Näkymän alueet, vasemmalta oikealle, tulee otsikoida nimillä Backlog, Working, Testing ja Approved. Tehtävänäkymästä tulisi löytyä painikkeet uuden tehtävän luomiseksi ja palaamiseksi projektinäkymään.

Luodut tehtävät tulisi näkyä kortteina oman vaiheensa mukaisella alueella. Jokaisessa tehtäväkortissa tulee lukea tehtävän nimi, tehtävän kuvaus, prioriteettitaso, vastuuhenkilö ja takaraja. Korteista olisi myös löydyttävä painikkeet tehtävän muokkaamiselle ja poistamiselle. Suunnitellun tehtävänäkymän luonnos esitetään kuvassa 23.




Kuva 23. Luonnos tehtävänäkymästä

Projekti- ja tehtävänäkymien painikkeet uuden projektin tai tehtävän luomiseksi, tai vanhan muokkaamiseksi, johtavat käyttäjän haluamaansa lomakkeeseen. Nämä lomakkeet tulisi toteuttaa siten, että niiden ulkoasu on yhteneväinen. Sovellusta on helpompi oppia käyttämään, kun kaikki lomakkeet toimivat samalla periaatteella.

Sovelluksen käyttöliittymän värimaailman valitsemiseen annettiin täysi toimintavapaus. Jotta tehtävienhallintajärjestelmän ulkoasu olisi yhteneväinen ja selkeä näkymästä toiseen, tulisi sovellus toteuttaa muutamalla päävärillä. Myös painikkeiden, listojen ja korttien sijoittelun tulisi olla ilmavaa ja tasapainoista, jotta sovelluksen käyttäminen olisi helppoa. Järkevällä sommittelulla sovellus saadaan myös miellyttävän näköiseksi.

## 3.3 Taustajärjestelmän suunnittelu

Toimeksiantajan vaatimuksesta taustajärjestelmä suunniteltiin toteutettavaksi Node.js-ajoympäristöllä ja Express-kehyksellä. Tärkein vaatimus taustajärjestelmälle oli käyttäjän henkilökohtaisen tunnisteen ja salasanan tietoturvallinen käsittely. Käyttäjätietojen turvallisen käsittelyn varmistamiseksi kehyksen väliohjelmiksi valittiin uuid, bcrypt ja JWT.

Uuid on väliohjelmisto, jolla luodaan samannimisiä uuid-tunnisteita. Uuid (Universal Unique Identifier) on 128-bittinen, yksilöllinen tunnistekoodi. Samassa tietokonejärjestelmässä luodut uuid-tunnisteet ovat uniikkeja.

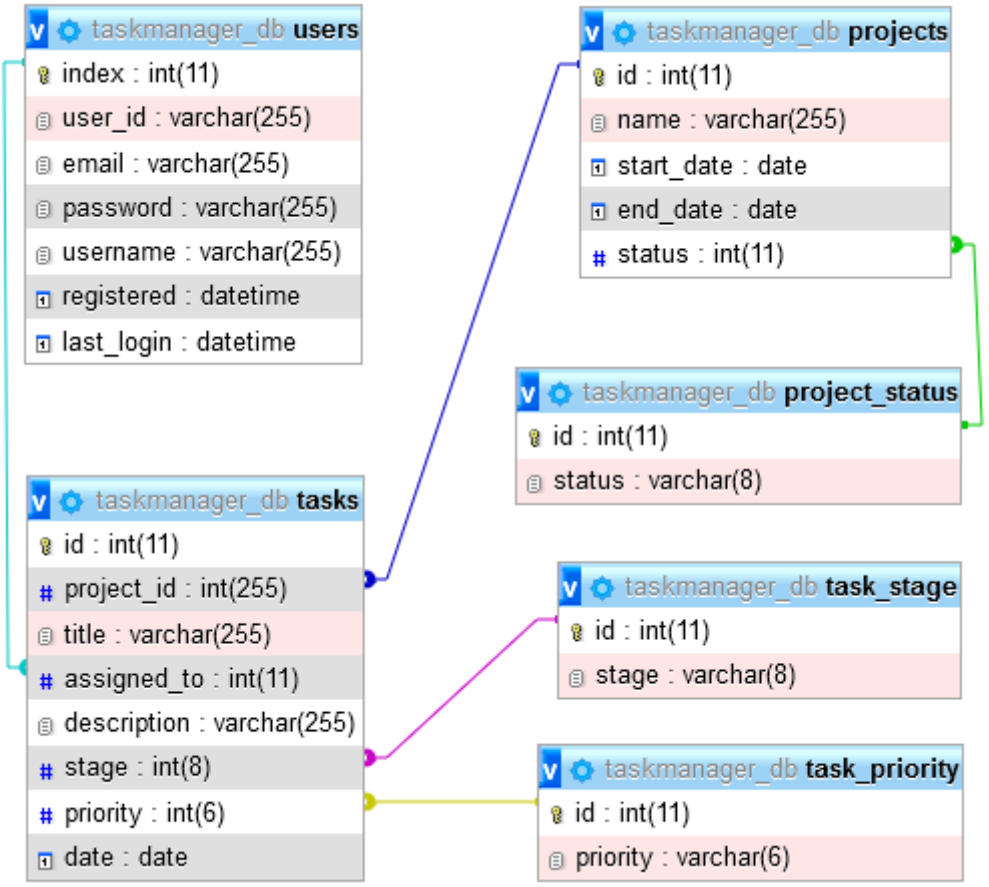
Bcrypt on tiivistealgoritmi, jota käytetään esimerkiksi salasanojen salaamiseen. Bcrypt perustuu Blowfish-algoritmille. Koska bcrypt käyttää suolausta ja tietokonejärjestelmän laskentatehon mukaan skaalautuvaa cost-parametriä, sitä pidetään tietoturvallisena salausmenetelmänä. Suolauksessa käytetään suola-arvoa salauksen satunnaistamiseen, ja cost-parametri hidastaa salauksen purkamista väsytyshyökkäyksellä.

JWT on lyhenne englannin kielen sanoista JavaScript Object Notation (JSON) Web Token. JWT on menetelmä, jota käytetään ohjelmistojen välisten käyttöoikeustietueiden hallinnoimiseen. JWT-käyttöoikeustietueiden ja salaisten avainten avulla palvelin voi tarkastaa, onko palvelimelle lähetetty pyyntö turvallinen. Tällä menetelmällä varmistetaan, että vain ne tahot, joilla on oikeudet taustajärjestelmän ja tietokannan tietoihin, pääsevät niihin käsiksi.

## 3.4 Tietokannan suunnittelu

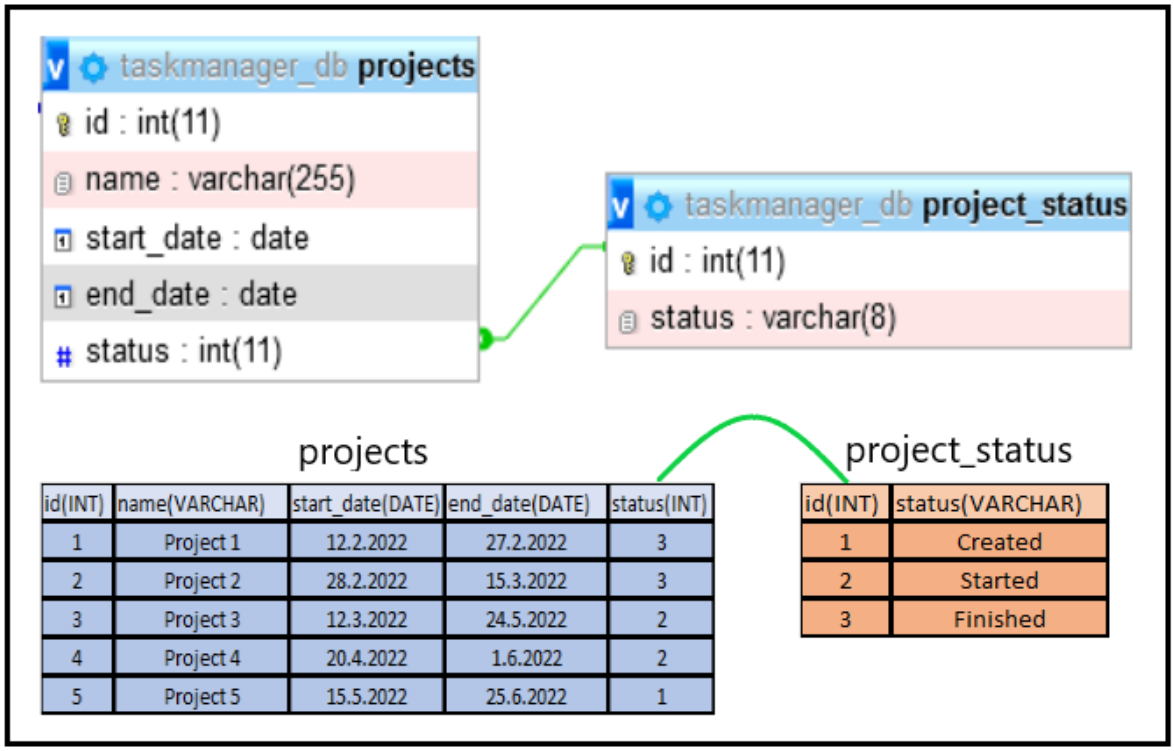
Projektille asetettujen vaatimusten perusteella tietokantaan suunniteltiin kuusi tietokantataulua: users (käyttäjät), projects (projektit), project_status (projektien tila), tasks (tehtävät),

task_priority (tehtävän prioriteettitaso) ja task_stage (tehtävän vaihe). Tietokannan suunniteltu rakenne taulujen välisine relaatioineen on havainnollistettu kuvassa 24.




Kuva 24. Suunniteltu tietokantarakenne

Tehtävienhallintajärjestelmän tietokannan suunnitteluvaiheessa päädyttiin ratkaisuun, jossa projektien tilat, tehtävien prioriteettitasot ja tehtävien vaiheet tallennetaan omiin tietokantatauluihinsa. Esimerkiksi projects-taulussa jokaisella projektilla on statussarakkeessa INT-tietotyypin luku. Project_status-taulussa, jolla on relaatio projects-tauluun, on INT-tyyppinen id-sarake, joka vastaa projects-taulun statuslukua. Project_status taulussa on lisäksi VARCHAR-tyyppinen sarake jokaisen tilan merkkijonomuotoiselle nimelle: Created, Started ja Finished. Projects- ja project_status-taulujen välille suunniteltu relaatio on nähtävissä kuvassa 25.




Kuva 25. Taulujen välinen relaatio

Sovelluksen käyttäjän avatessa käyttöliittymässä projektien muokkausnäkymän, projektin mahdolliset tilavaihtoehdot haetaan tietokannasta oliomuodossa. Jokainen olio sisältää INT-tietotyyppisen luvun ja merkkijonomuotoisen nimen, esimerkiksi: {”id”: 1, ”status”: ”Created”}. Kun projektien muokkausnäkymä avataan, valikot täytetään olioiden selkokielisillä merkkijononimillä muokkausten ymmärrettävyyden parantamiseksi. Kun käyttäjä vaihtaa projektin tilan Created-tilasta Started-tilaan ja sovellus lähettää muokkauspyynnön taustajärjestelmälle, pyyntö sisältää vain valitun olion INT-tietotyyppisen luvun.

Vähentämällä merkkijonomuotoisia pyyntöjä taustajärjestelmälle vähennetään samalla SQL-injektioiden riskiä. SQL-injektio on hakkerointimenetelmä, jossa käytetään hyväksi verkkosivustojen ja web-sovellusten syötekenttien haavoittuvuuksia.

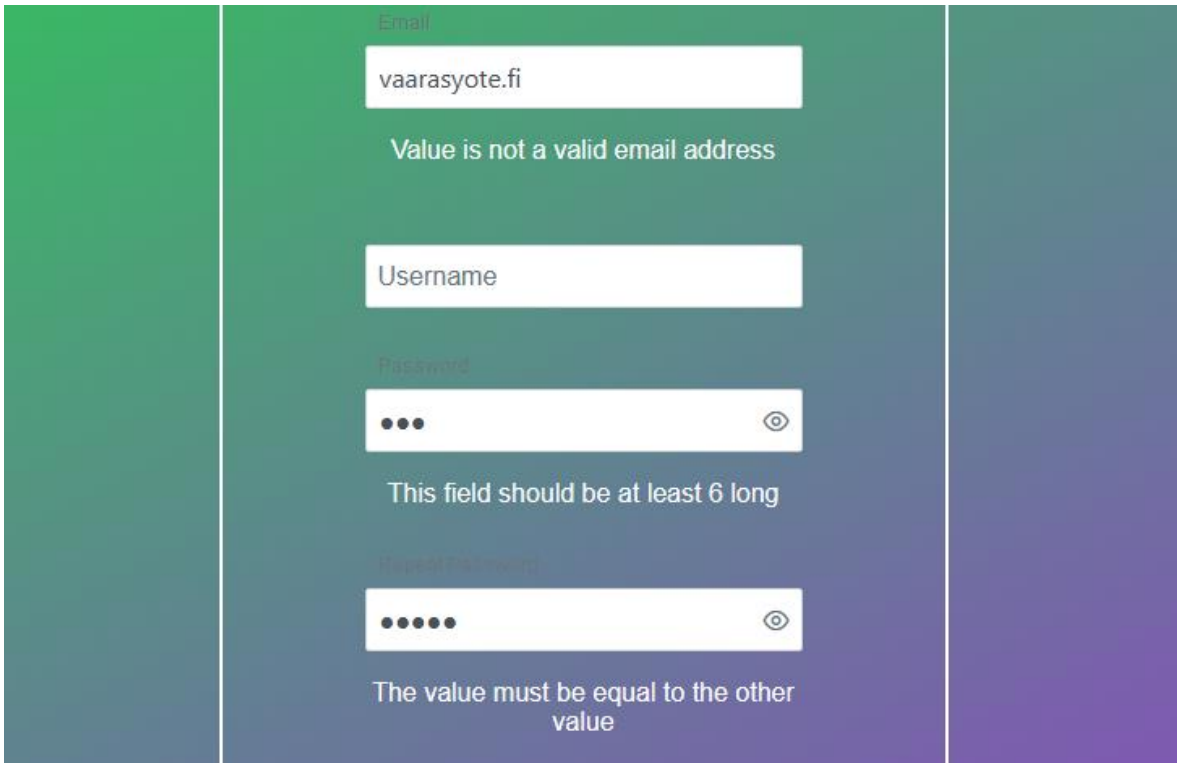
Valittu tietokantarakenne tehostaa myös muutosten tekoa tulevaisuudessa. Kun projektien tiloja, tehtävien prioriteettitasoja ja tehtävien vaiheita käsitellään pyynnöissä lukuina, ja lukujen merkkijonomuotoiset vastineet ladataan tietokannasta käyttöliittymän käyttöön, vältetään näiden tietojen koodaaminen erikseen kaikkiin niitä tarvitseviin näkymiin. Jos myöhemmin halutaan esimerkiksi muuttaa projektin tila Started muotoon Initiated, muutos tehdään vain yhteen tietokantatauluun, eikä jokaiseen käyttöliittymänäkymään.

# 4 Tehtävienhallintajärjestelmän toteutus

## 4.1 Käyttöliittymä

Vuen ja PrimeVuen lisäksi käyttöliittymän toteutuksessa käytettiin PrimeFlex-apukirjastoa, jolla luotiin käyttöliittymänäkymien rakenne. PrimeVue-komponenttien lisäksi käyttöliittymän toteutuksessa käytettiin itse tehtyjä komponentteja ja apukirjaston ulkopuolista, mutta sen kanssa yhteensopivaa FullCalendar-komponenttia.

Käyttöliittymän syötekentissä käytettiin Vuelidate-apukirjastoa syötteiden oikeellisuuden tarkastamiseen. Vuelidaten avulla varmistettiin, ettei lomakkeita voida lähettää taustajärjestelmälle, jos tarvittavat syötteet ovat väärässä muodossa tai tyhjiä. Esimerkki Vuelidaten virheilmoituksesta väärän syötteen yhteydessä on esitetty kuvassa 26.

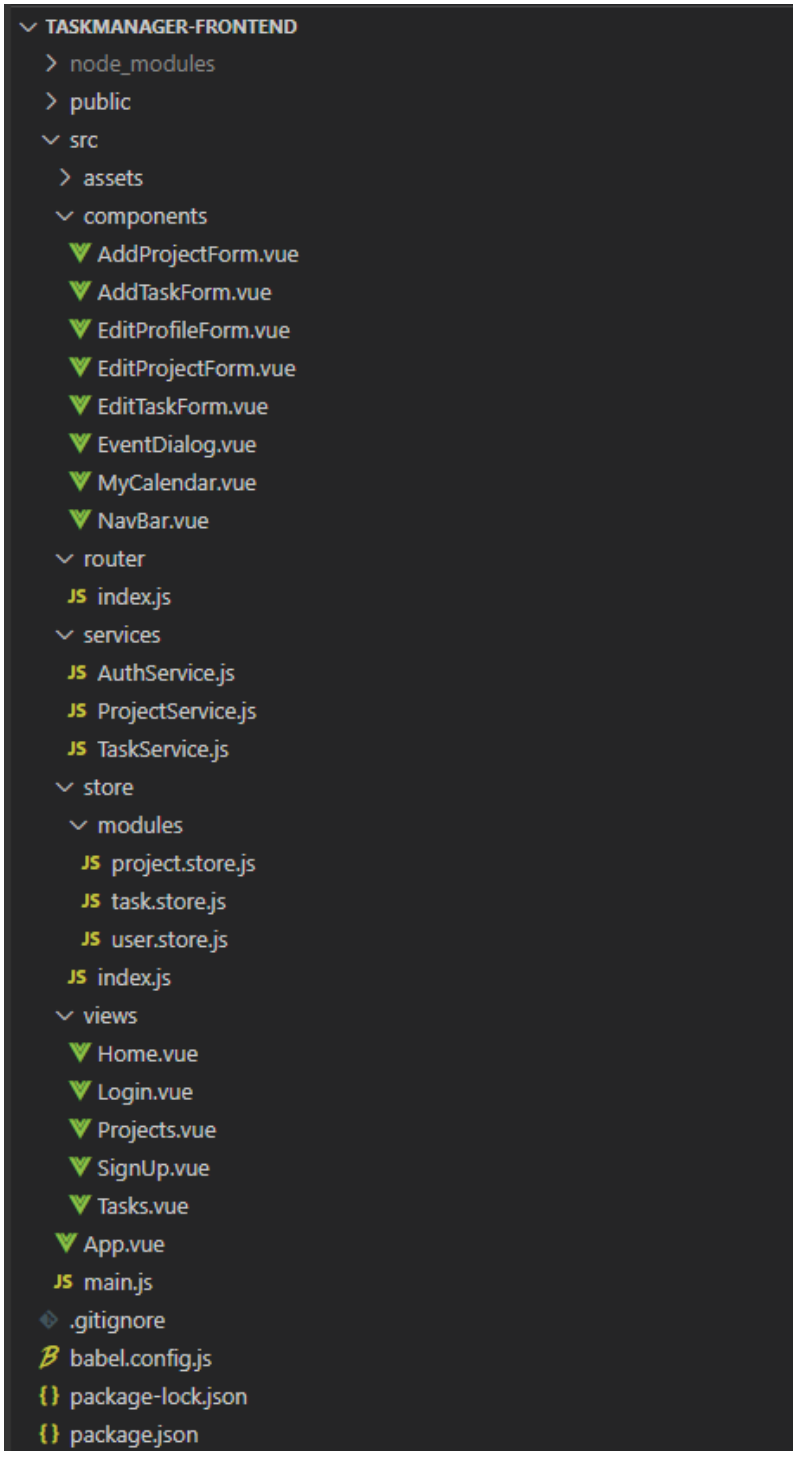



Kuva 26. Vuelidaten virheilmoitukset vääristä syötteistä

### 4.1.1 Rakenne

Käyttöliittymän koodirakenne toteutettiin noudattaen Vuex-apukirjaston store-mallia. Koodin lukemisen ja sovelluksen ylläpitämisen tehostamiseksi luotiin useampi store-tiedosto; ”user.store.js” käyttäjän tilojen hallintaan, ”project.store.js” projektien tilojen hallintaan ja

”task.store.js” tehtävien tilojen hallintaan. Käyttäjän näkymien välisen liikkumisen hallitsemiseksi käytettiin Vue Router-apukirjastoa. Tehtävienhallintajärjestelmän tiedosto- ja kansiorakenne on nähtävissä kuvassa 27.

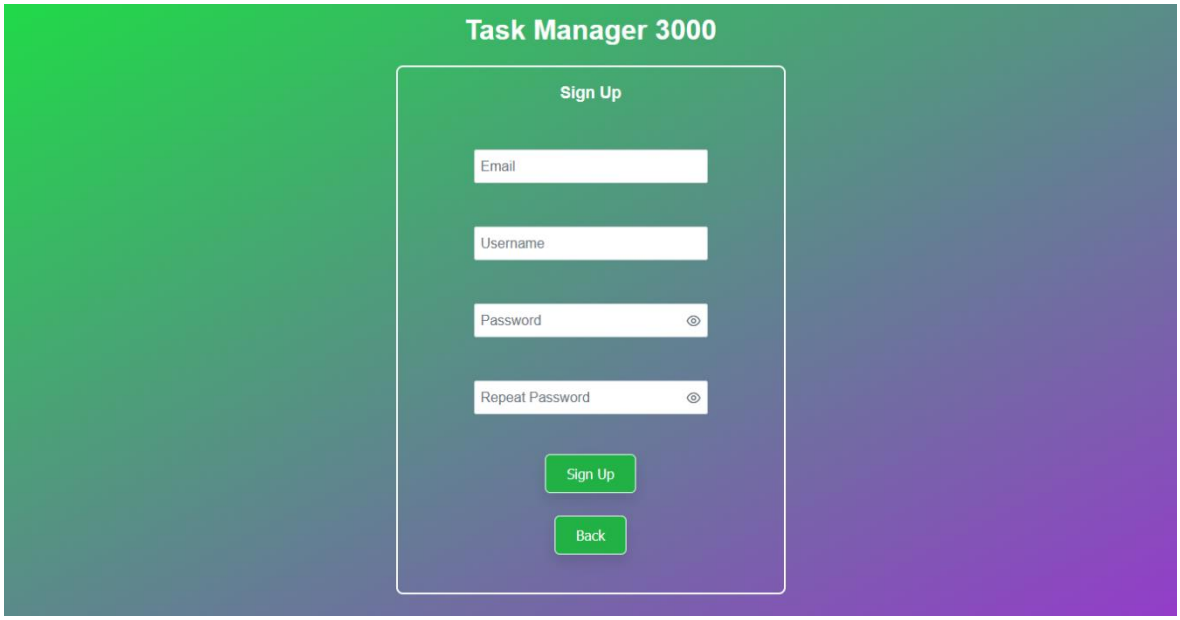



Kuva 27. Käyttöliittymän tiedostot

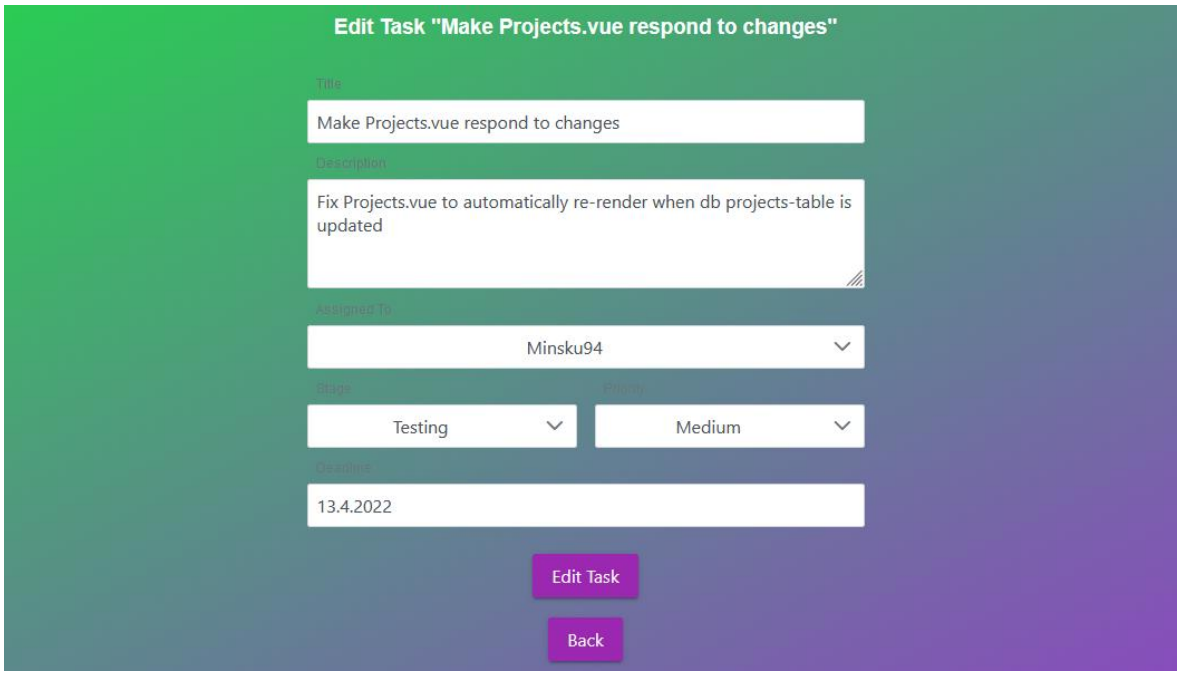
Tehtävienhallintajärjestelmän ylläpitämisen sujuvoittamiseksi luotiin service-tiedostot. Service-tiedostojen tehtävänä on vastaanottaa store-tiedostoissa tehdyt funktiokutsut ja tehdä niiden perusteella taustajärjestelmäkutsut. Taustajärjestelmän kutsumiseen käytettiin Axios-apukirjastoa.

### 4.1.2 Ulkoasu

Käyttöliittymän ulkoasu tehtiin selkeäksi ja helppokäyttöiseksi. PrimeVue-apukirjastosta otettiin käyttöön ne komponentit, joita tarvittiin järjestelmän toteuttamiseen. Lomakenäkymissä käytettiin esimerkiksi TextInput-, Email-, Password-, Calendar- ja DropDown-komponentteja. Järjestelmän rekisteröitymis- ja tehtävänmuokkauslomakkeet ovat nähtävissä kuvissa 28 ja 29.

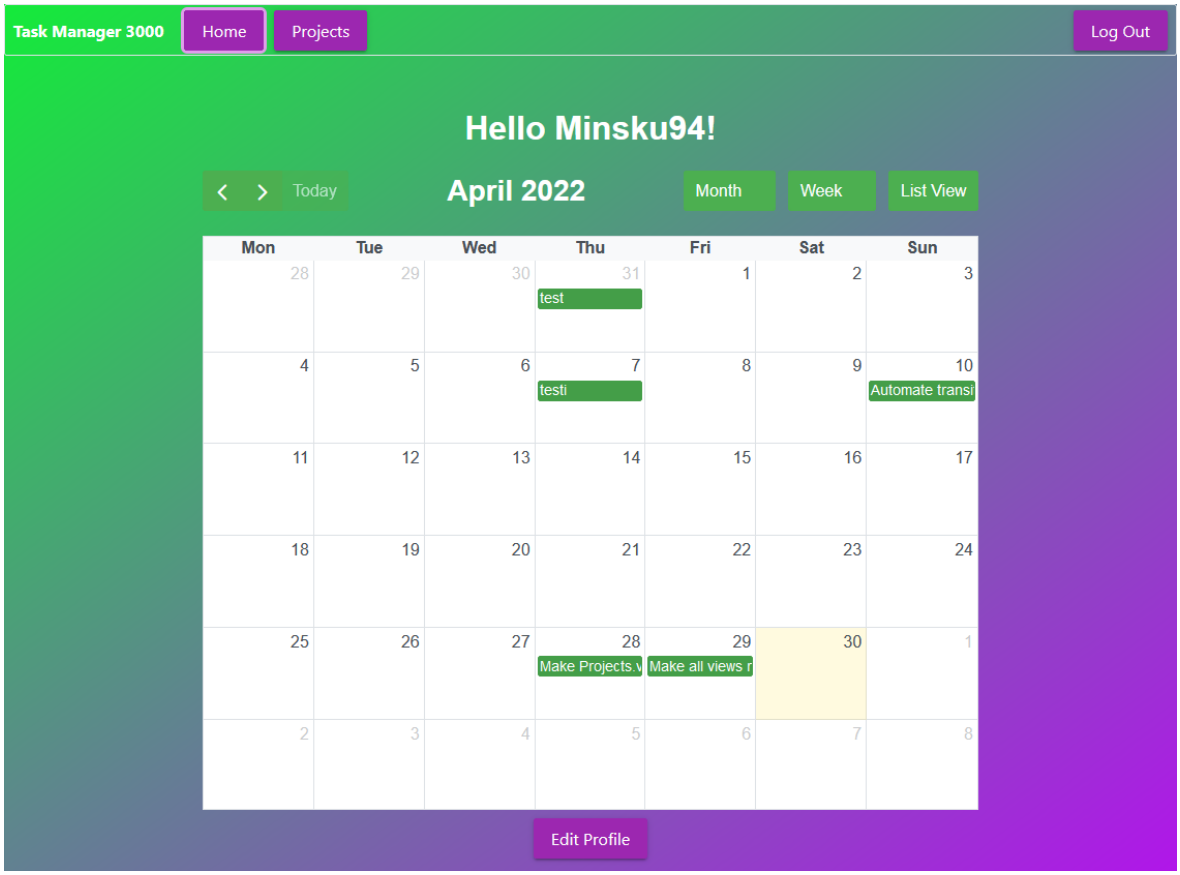



Kuva 28. Rekisteröitymislomake

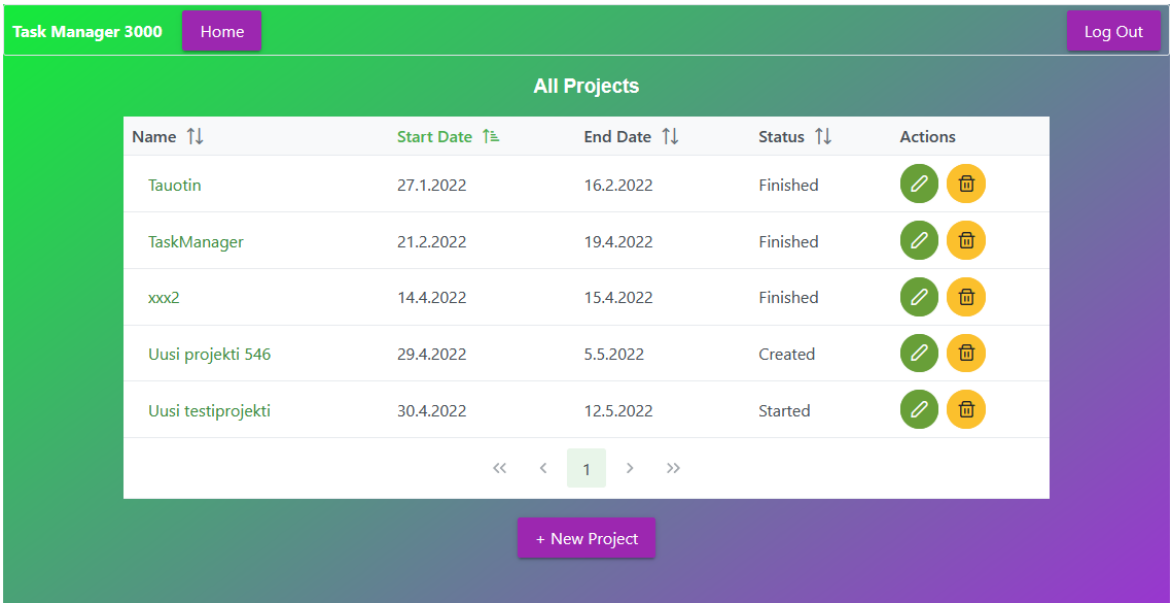



Kuva 29. Tehtävänmuokkauslomake

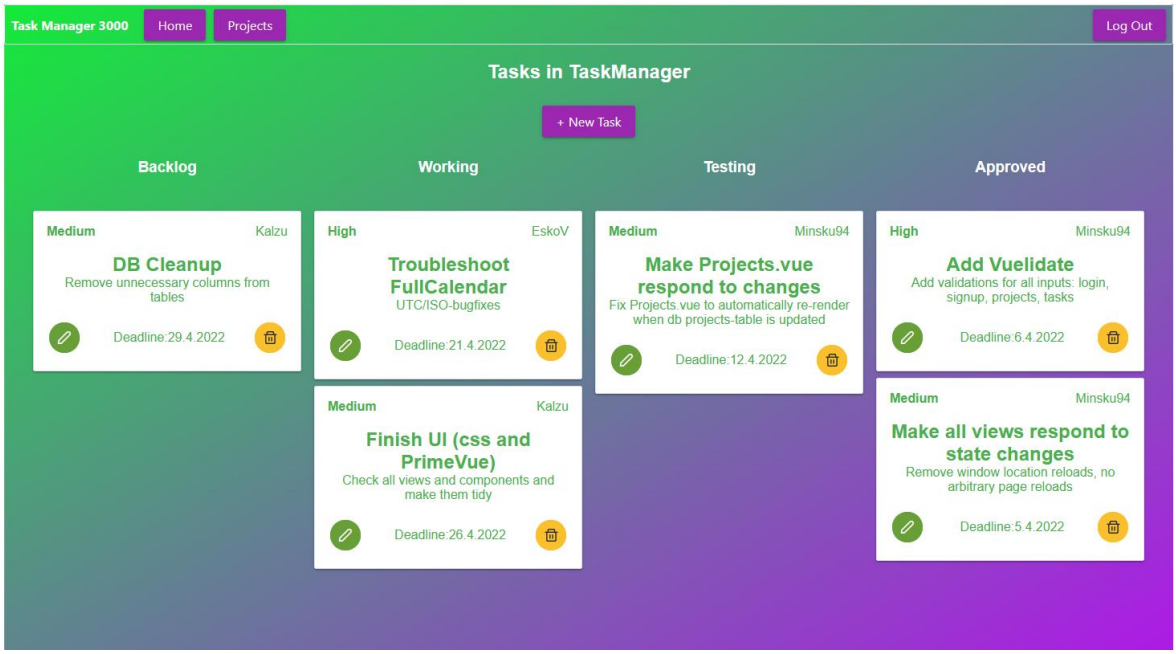
Käyttäjän kotinäkymässä käytettiin PrimeVuen kanssa yhteensopivaa FullCalendar-komponenttia käyttäjän tehtävien takarajojen havainnollistamiseksi. Projekti- ja tehtävänäkymissä käytettiin muun muassa DataTable- ja Card-komponentteja. Koti-, projekti- ja tehtävänäkymät on esitetty kuvissa 30, 31 ja 32.




Kuva 30. Kotinäkymä



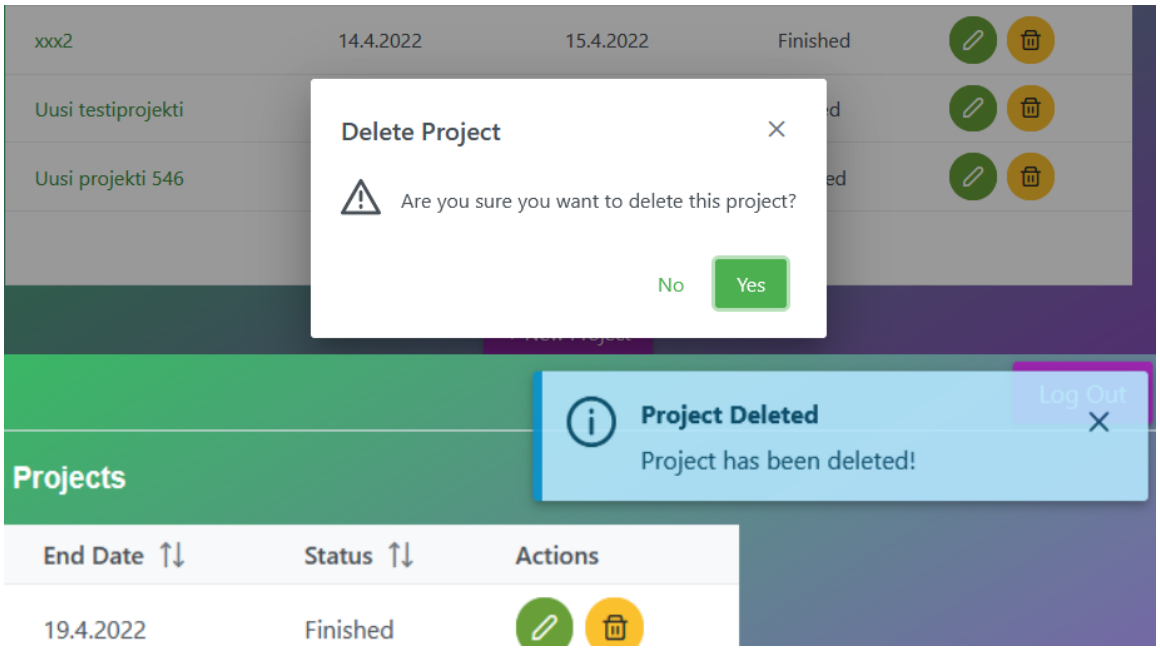

Kuva 31. Projektinäkymä




Kuva 32. Tehtävänäkymä

Komponenttien asetteluun ja näkymien sommitteluun käytettiin PrimeFlex-apukirjastoa. PrimeFlexin grid-järjestelmän avulla näkymä voidaan jakaa tietyn kokoisiin alueisiin. Käyttöliittymäkomponentit asetetaan PrimeFlex-alueiden sisälle. Jos komponentin sijoittelua haluaa muuttaa, muutokset tehdään alueen asetuksiin.

Käyttöliittymän toteutuksessa huomioitiin vahinkoklikkaukset estämällä tehtävien ja projektien poistaminen yhdellä napinpainalluksella. Projektien ja tehtävien yhteydestä löytyviä roskakori-ikoneita klikatessa käyttäjää pyydetään varmistamaan valintansa. Valinnan varmistamisen tai perumisen jälkeen käyttäjälle ilmoitetaan tehdystä valinnasta. Projektin poistamisen varmistuskysely ja sen jälkeinen ilmoitus on nähtävissä kuvassa 33.




Kuva 33. Varmistuskysely ja ilmoitus

## 4.2 Taustajärjestelmä ja tietokanta

Taustajärjestelmä toteutettiin Express-kehyksellä. Käyttäjätietojen tietoturvallinen käsittely toteutettiin väliohjelmistojen avulla. Eri verkkotunnusten välisten resurssipyyntöjen sallimiseksi käytettiin cors-väliohjelmistoa (Cross-Origin Resource Sharing).

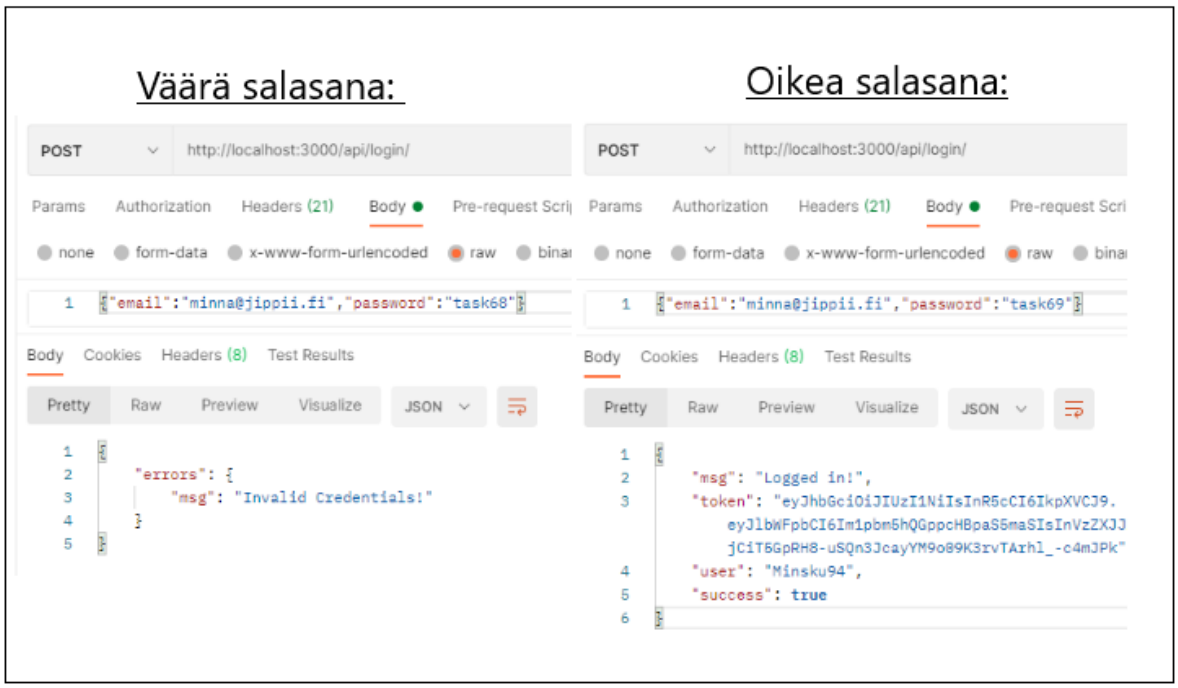
Toteutetussa taustajärjestelmässä käytettiin Express-kehyksen Router-funktiota. Router-funktiota käyttäessä URL-reittien käsittelyfunktiot sijoitetaan Express-sovelluksen päätiedoston ulkopuolelle omaan tiedostoonsa. Täten koodia on helpompi lukea, kun päätiedosto sisältää vain koko sovellusta käsittelevät asiat. Mikäli sovellusta laajennetaan myöhemmin, tämä koodirakenne tekee ohjelmistorajapintojen lisäämisestä mutkatonta. Myös taustajärjestelmän ylläpitäminen selkeytyy.

Tehtävienhallintajärjestelmän tietokanta toteutettiin alkuperäisen suunnitelman mukaisesti. Toteutetussa MariaDB-tietokannassa säilytetään sovelluksen käyttäjä-, tehtävä- ja projektitiedot. Tietokannan toteutuksessa huomioitiin käyttäjien tunnisteiden ja salasanojen tietoturvallinen säilöntä tallentamalla tiedot tietokantatauluun salatussa muodossa.

## 4.3 Testit

Tehtävienhallintasovelluksen taustajärjestelmän ohjelmistorajapintojen toiminta testattiin Postman-sovelluksella. Jotta voitiin testata kirjautumisen takana olevat rajapinnat, kopioitiin testejä varten luodun käyttäjän kirjautumisen HTTP-POST-pyyntö Postman-sovellukseen.

Kirjautumisen toiminta testattiin erilaisilla yhdistelmillä vääriä ja oikeita sähköpostiosoitteita ja salasanoja. Kirjautuminen onnistui vain, mikäli pyyntö sisälsi tietokantaan rekisteröityneelle käyttäjälle tallennetun sähköpostiosoitteen ja salasanan. Testattu rajapinta, testipyyntöjen sisällöt ja taustajärjestelmän vastaukset sekä väärälle että oikealle salasanalle on nähtävissä kuvassa 34.




Kuva 34. Kirjautumisen testitulokset Postman-sovelluksessa

Kaikki taustajärjestelmän API:t testattiin. Ohjelmistorajapintoja oli yhteensä 16 kappaletta. Testeillä varmistuttiin siitä, ettei sovelluksen taustajärjestelmän tai tietokannan tietoihin pääse käsiksi ilman voimassa olevaa käyttäjätunnusta ja salasanaa.

Sovellusta testattiin myös Firefox-verkkoselaimella. Osoitekenttään syötettiin reittejä sovelluksen koti-, tehtävä- ja projektinäkymiin ilman sisäänkirjautumista. Sovellus toimi halutulla tavalla, siirtäen kirjautumattoman käyttäjän kaikista kirjautumisen vaativista reiteistä kirjautumisnäkymään. Lisäksi tehtävienhallintajärjestelmän toimintaa testattiin Google Chrome-

ja Microsoft Edge -selaimilla. Sovelluksen toiminnassa ei havaittu eroavaisuuksia selainten välillä.

# 5 Yhteenveto ja jatkokehitys

Tämän työn tavoitteena oli toteuttaa tehtävienhallintajärjestelmä web-sovelluksena. Päätavoitteina oli käyttäjän rekisteröitymisen, kirjautumisen, tehtävän- ja projektinhallinnan toteutus. Muita tavoitteita oli käyttäjäystävällisen käyttöliittymän toteuttaminen ja käyttäjän tietojen tietoturvallinen käsitteleminen.

Työn tavoitteet saavutettiin, ja toimeksiantaja oli tyytyväinen lopputulokseen. Tehtävienhallintajärjestelmän käyttöliittymä, taustajärjestelmä ja tietokanta toteutettiin toimeksiantajan määrittelemien vaatimusten mukaisesti. Käyttöliittymä toteutettiin toimeksiantajan toiveiden mukaisilla apukirjastoilla, ja tehtävienhallintajärjestelmään sisällytettiin vaaditut tehtävien- ja projektienhallintatoiminnallisuudet.

Käyttäjätietojen tietoturvallinen käsittely varmistettiin käyttämällä taustajärjestelmässä bcrypt- ja JWT-väliohjelmistoja. Käyttäjäystävällisyyteen panostettiin luomalla mahdollisimman yksinkertainen käyttöliittymä. Käyttäjäkokemuksen parantamiseksi sovellukseen lisättiin kotinäkymä, jossa käyttäjä voi muokata tietojaan ja seurata omien tehtäviensä takarajoja sekä kalenterista että listalta.

Tehtävienhallintajärjestelmän toteutuksessa huomioitiin sovelluksen mahdollinen jatkokehittäminen. Käyttöliittymän modulaarinen rakenne ja taustajärjestelmän Router-funktion hyödyntäminen sallivat lisätoiminnallisuuden vaivattoman lisäämisen.

Sovelluksen jatkokehittämiseksi on monta ideaa. Käyttäjien, tehtävien ja projektien lisäksi järjestelmässä voisi olla tiimejä, rooleja ja oikeuksia. Käyttäjät voisivat kuulua tiimeihin, ja tiimit voisi liittää projekteihin. Käyttäjillä voisi olla erilaisia rooleja tiimien sisällä, ja roolit voisivat puolestaan määritellä, millaisia oikeuksia käyttäjällä on muokata projekteja ja tehtäviä.

Sovelluksen Tasks-näkymästä voisi tehdä käyttäjäystävällisemmän tekemällä tehtäväkorteista hiirellä liikutettavia. Työn toteutusvaiheessa PrimeVue-kirjastosta ei löytynyt tilanteeseen sopivaa ”drag-n-drop”-toimintoa, mutta tilanne saattaa muuttua tulevien apukirjastopäivitysten myötä. Käyttäjäystävällisyyttä voisi myös parantaa optimoimalla sovellusnäkymät mobiililaitteiden näytöille.

Yllä mainittujen jatkokehitysideoiden lisäksi tehtävienhallintajärjestelmään voisi toteuttaa viestijärjestelmän, jolla käyttäjät voisivat jättää kommentteja ja esittää kysymyksiä toisilleen. Viesteistä voisi olla hyötyä tehtäviin liittyvien epäselvyyksien ratkaisemisessa. Viestijärjestelmä mahdollistaisi myös käyttäjille lähetettävät automatisoidut ilmoitukset esimerkiksi tehtävän takarajan lähestyessä.

Toteutettu tehtävienhallintajärjestelmä soveltuu sovelluskehitysprojektien ja niihin kuuluvien tehtävien hallintaan. Sitä voi käyttää myös kalenteri- ja muistiosovelluksena arjen ajanhallinnan helpottamiseksi.

## Lähteet

Ahuja, K., Chandra, V., Lord, V. & Peens, C. 2021. Ordering In: The rapid evolution of food delivery. MacKinsey & Company. 22.9.2021. Viitattu 20.3.2022. Saatavissa https://www.mckinsey.com/industries/technology-media-and-telecommunications/our-insights/ordering-in-the-rapid-evolution-of-food-delivery

AndreKR. 2017. Framework/strategy for deeply nested data input. Viitattu 6.4.2022. Saatavissa https://forum.vuejs.org/t/framework-strategy-for-deeply-nested-data-input/19576

Chaitanya, C. 2020. Two-way data binding with v-model. Viitattu 5.4.2022. Saatavissa https://javabeginnerstutorial.com/vue-js/14-two-way-binding-v-model/

Cromwell, V. 2016. Evan You. Between the Wires. 3.11.2016. Viitattu 26.3.2022. Saatavissa https://web.archive.org/web/20170603052649/https://betweenthewires.org/2016/11/03/evan-you/

Expressjs a. Hello world example. Viitattu 6.4.2022. Saatavissa https://expressjs.com/en/starter/hello-world.html

Expressjs b. Writing middleware for use in Express apps. Viitattu 6.4.2022. Saatavissa https://expressjs.com/en/guide/writing-middleware.html

Expressjs c. Using middleware. Viitattu 6.4.2022. Saatavissa https://expressjs.com/en/guide/using-middleware.html

IBM Cloud Education. 2020. Three-Tier Architecture. 28.10.2020. Viitattu 26.3.2022. Saatavissa https://www.ibm.com/cloud/learn/three-tier-architecture

Imsirovic, A. 2018. Vue.js Quick Start Guide. Viitattu 5.4.2022. Saatavissa https://subscription.packtpub.com/book/front-end-web-development/9781789344103/3/ch03lvl1sec23/the-structure-of-our-vue-cli-based-project

Kuuki Marketing Lab Oy. 2022. Kuukin tarina. www.kuuki.fi. Viitattu 5.4.2022. Saatavissa https://www.kuuki.fi/kuuki/kuukin-tarina/

Obrizan, A. 2020. Benefits of JavaScript on the Server. LeanyLabs-blogi 21.6.2020. Viitattu 7.5.2022. Saatavissa https://leanylabs.com/blog/benefits-of-javascript-on-the-server/

Rice, C. 2000. Different Types of Software Installation. Server Watch. 24.12.2000. Viitattu. 26.3.2022. Saatavissa https://www.serverwatch.com/guides/different-types-of-software-installation/

Shcheglov, M. 2019. MVVM Design Pattern with Combine framework on iOS. Viitattu 5.4.2022. Saatavissa https://medium.com/@mshcheglov/mvvm-design-pattern-with-combine-framework-on-ios-5ff911011b0b

Stack Overflow. 2021. 2021 Developer Survey. Viitattu 5.4.2022. Saatavissa https://insights.stackoverflow.com/survey/2021#technology-most-popular-technologies

Statista. 2022. Most used web frameworks among developers worldwide, as of 2021. Viitattu 5.4.2022. Saatavissa https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web/

Techopedia. 2017. Web-Based Application. www.techopedia.com. 21.6.2017. Viitattu 26.3.2022. Saatavissa https://www.techopedia.com/definition/26002/web-based-application

Vue CLI a. Installation. Viitattu 7.5.2022. Saatavissa https://cli.vuejs.org/guide/installation.html

Vue CLI b. Creating a Project. Viitattu 7.5.2022. Saatavissa https://cli.vuejs.org/guide/creating-a-project.html

Vuejs a. Quick Start – Without Build Tools. Viitattu 6.4.2022. Saatavissa https://vuejs.org/guide/quick-start.html#without-build-tools

Vuejs b. Quick Start – With Build Tools. Viitattu 6.4.2022. Saatavissa https://vuejs.org/guide/quick-start.html#with-build-tools

Vuexjs. What is Vuex? Viitattu 15.4.2022. Saatavissa https://vuex.vuejs.org/#what-is-a-state-management-pattern

W3Schools. What is Full Stack? Viitattu 7.5.2022. Saatavissa https://www.w3schools.com/whatis/whatis_fullstack.asp