Au Nguyen

Web-sovellus asiakastietojen hakemiseksi

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

11.4.2023

Tiivistelmä

Tekijä:	Au Nguyen
Otsikko:	Web-sovellus asiakastietojen hakemiseksi
Sivumäärä:	50 sivua
Aika:	11.4.2023
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Ohjelmistotuotanto
Ohjaajat:	Lehtori Vesa Ollikainen
	CTO Antti Marjala

Opinnäytetyön toimeksiantajana toimi Apix Messaging Oy -yritys, joka on laskujen ja palkkalaskelmien välityspalveluihin sekä tilaus- ja toimitusketjujen automatisointiin liittyvien sanomanvälityspalveluiden suomalainen uranuurtaja.

Opinnäytetyön tavoitteena oli luoda modernin näköinen web-sovellus, jolla pystytään hakemaan, poistamaan ja muokkaamaan yrityksen asiakkaiden tietoja. Sovellus on tarkoitettu ainoastaan yrityksen sisäiseen käyttöön.

Opinnäytetyön sivutavoitteena oli kehittää tekijän osaamista ja oppia uusia tekniikoita. Teoriaosuudessa käytiin läpi projektissa käytetyt teknologiat, kuten Go, Vue.js, Vuex, MySQL, ja kerrottiin, miten niitä käytettiin projektissa.

Opinnäytetyön tuloksena saatiin toimiva web-sovellus yrityksen asiakkaiden etsimistä varten. Sovellus täyttää asetetut tavoitteet ja tuottaa uudelleenkäytettäviä komponentteja ja koodipohjaa jatkokehitykseen. Opinnäytetyön sivutuloksena saatiin lisää tietoa ja kokemusta käytetyistä tekniikoista sekä opittiin tehokkaampia tapoja ohjelmoida.

Avainsanat: Vue.js, Go, Hakutyökalu

Abstract

Author:	Au Nguyen
Title:	Web Application for Retrieving Customer Information
Number of Pages:	50 pages
Date:	11 April 2023
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communications Technology
Professional Major:	Software Engineering
Supervisors:	Vesa Ollikainen, Senior Lecturer
	Antti Marjala, Chief Technical Officer

The aim of the study was to create a modern looking web application for Apix Messaging Oy company. The web application was to be able to retrieve, delete and modify the company's customer data. The application is for intra-corporate use only.

The theoretical part examines the technologies used in the project, such as Go, Vue.js, Vuex, Mysql and explains how they were used in the project.

The result is a functional web application for the search of the company's customers. The application meets the set objectives and produces reusable components and code base for further development. The project also resulted in the author gaining more knowledge and experience of the techniques used and to learn more effective ways of programming.

Keywords: Vue.js, Go, Search tool

Sisällys

Lyhenteet

Bash: *Bourne-Again Shell*. Unix-pohjainen komentotulkin ohjelmointikieli.

CORS: *Cross-Origin Resource Sharing*. Web-sovellusten suojausmekanismi, joka auttaa estämään tietoturva-aukkoja.

HTML: *Hypertext Markup Language*. Standardoitu merkintäkieli web-sivujen rakentamiseen.

JSON: *JavaScript Object Notation* on yksinkertainen tiedonsiirtomuoto, joka perustuu JavaScript-kielen syntaksiin.

NPM: *Node Package Manager*. Avoimen lähdekoodin pakettien hallintajärjestelmä JavaScript-ohjelmointikielelle.

RDBMS: *Relational Database Management System*. Tietokantaohjelmisto, joka käyttää relaatiomallia tietojen tallentamiseen ja hallintaan.

SFC: *Single-File Component*. Yksittäinen tiedosto, joka sisältää kaikki malli-, näkymä- ja logiikkakoodit yhdessä tiedostossa.

TLS: *Transport Layer Security*. Salausprotokolla, joka auttaa suojaamaan tietoliikennettä verkon yli.

URL: *Uniform Resource Locator*. Osoite, joka tunnistaa resurssin verkkopalvelimella.

1 Johdanto

Tämä opinnäytetyö käsittelee asiakkaiden tietojen hakutyökalun web-sovelluksen suunnittelua ja toteutusta. Tavoitteena oli luoda käyttäjäystävällinen ja tehokas hakutyökalu, joka helpottaa asiakastietojen etsimistä ja hallintaa. Hakuominaisuudet ovat tärkeä osa asiakastietojärjestelmiä, sillä ne mahdollistavat asiakastietojen nopean ja tehokkaan etsimisen. Se myös vähentää manuaalista työtä ja vähentää mahdollisuutta virheisiin. Opinnäytetyön sivutavoitteena oli kehittää tekijän osaamista ja oppia uutta tekniikkaa.

Sovellus tarjoaa useita erilaisia hakuvaihtoehtoja, kuten hakua asiakastunnusnumeron, asiakasnumeron, puhelinnumeron tai sähköpostiosoitteen perusteella. Lisäksi sovellus tarjoaa mahdollisuuden lisätä, muokata ja poistaa asiakastietoja.

Opinnäytetyön toimeksiantajana toimi Apix Messaging Oy -yritys, joka tarjoaa laskujen ja palkkalaskelmien välityspalveluja sekä tilaus- ja toimitusketjujen automatisointiin liittyviä sanomanvälityspalveluja. Yritys on perustettu 2010, ja se on Suomen verkkolaskumarkkinoiden nopeimmin kasvava operaattori. Apixilla on jo yli 12 000 asiakasta, ja vahva kasvu jatkuu edelleen. (1.)

Luvussa 2 tarkastellaan tarvetta asiakastietojen hakutyökalulle sekä esitellään ratkaisu ja tavoite. Luvussa 3 käydään läpi projektissa käytetyt teknologiat, kuten Go, Vue.js, Vuex, Mysql ja kerrotaan, miten niitä käytetään projektissa. Luvussa 4 käsitellään sovelluksen arkkitehtuuria ja kansiorakenteita. Luvussa 5 käsitellään sovelluksen rakennuksen eri vaiheita ja jokainen sovelluksen näkymä tarkastellaan yksityiskohtaisemmin. Luvussa 6 kerrotaan saadut tulokset ja arvioidaan sovelluksen toteutuksesta.

Opinnäytetyön tuloksena saatiin toimiva web-sovellus yrityksen asiakkaiden etsimistä varten. Sovellus täyttää asetetut tavoitteet ja tuottaa

uudelleenkäytettäviä komponentteja ja koodipohjaa jatkokehitykseen. Opinnäytetyön sivutuloksena saatiin lisää tietoa ja kokemusta käytetyistä tekniikoista sekä opittiin tehokkaampia tapoja ohjelmoida. Opinnäytetyön lukemalla lukija saa käsityksen siitä, kuinka Go-ohjelmointikieltä voidaan käyttää ohjelmistokehitysprojektissa, sekä sen ominaisuuksista, eduista ja haasteista verrattuna muihin ohjelmointikieliin.

2 Lähtötilanne

Yrityksellä oli aiemmin Perl-ohjelmointikielellä kirjoitettu skripti, jonka avulla asiakastiedot näytetään terminaali-ikkunassa. On kyllä olemassa tietty työkalu, minkä takia tarvitaan uusi selainsovellus ?

Tarkastellaan skenaariota, jossa myyjä A tarvitsee asiakkaan X puhelinnumeron aloittaakseen yhteydenoton uutta projektia varten, kun taas markkinointihenkilöstö B tarvitsee asiakkaan sähköpostiosoitteen lähettääkseen ilmoituksia palvelun hinnankorotuksista. Vastaavasti taloushallinon työntekijä C tarvitsee asiakkaan sähköpostitiedot muistutuslaskujen lähettämistä varten. Nämä työntekijät kohtaavat saman ongelman, sillä heillä ei ole pääsyä tietokantaan, eivätkä he osaa käyttää Linux-terminaalia. Lisäksi terminaali-ikkunassa näytetyt tiedot eivät olleet visuaalisesti houkuttelevia ja käyttäjäystävällisiä, mikä johti merkittävään tehokkuuden vähenemiseen. Edellä mainituista syistä kehitetiin uusi asiakastietojen hakutyökalu, joka on visuaalisesti houkutteleva ja käyttäjäystävällinen ja jonka avulla kaikille organisaation jäsenille on helpompi päästä käsiksi tarvittaviin tietoihin.

Selainsovellus on käytettävissä sekä tietokoneella että puhelimella, mikä tekee siitä erittäin kätevän ja käytännöllisen.

Toinen yrityksen tarve oli, että kehittäjä ennen projektin toteuttamista ei ole koskaan työskennellyt Go-ohjelmointikielen kanssa eikä hänellä ole todellista kokemusta palvelinpuolella työskentelemisestä. Siksi projektikehityksen kautta yritys oli luonut kehittäjälle edellytykset oppia ja tutustua Go-ohjelmointikieleen,

ja samalla kehittäjä pääsee käsiksi työkaluihin ja tehtyihin projekteihin tällä kielellä.

3 Käytetyt tekniikat

Valittaessa teknologioita selainpuolen kehittämiseen projektissa tulee täyttää kriteerit, kuten helppokäyttöisyys / dokumentaatio, kehittäjien saatavuus, työn määrä, laaja yhteisö / yhteisön tuki.

Taulukossa 1 vertaillaan Reactia, Angularia ja Vuea annettujen kriteerien perusteella. Luokitellaan kriteerit asteikolla 1-10. Arvioinnit ovat yleisluonteisia, ja ne perustuvat tiimin kehittäjien henkilökohtaisiin käyttökokemuksiin.

Taulukko 1. Reactin, Angularin ja Vuen vertailu.

Kriteeri	**React**	**Angular**	**Vue**
Helppokäyttöisyys / Dokumentaatio	9	8	10
Laaja yhteisö / Yhteisön tuki	10	9	10
Kehittäjien saatavuus	8	7	10
Työn määrä	9	8	10

Reactilla on erinomainen dokumentaatio, ja sen syntaksi on helppo ymmärtää. Sekä Reactin ja Vuen syntaksi on helppokäyttöinen. Angularilla on myös hyvä dokumentaatio, mutta sen syntaksi on vähemmän intuitiivinen ja sitä voi olla vaikeampi oppia. Vuen dokumentaatio on kattavaa, ja se sisältää paljon esimerkkejä ja selityksiä. (2.)

Reactilla ja Vuellä on laaja ja aktiivinen yhteisö, joka tarjoaa paljon tukea ja apua käyttäjille. Näiden kehysten yhteisöt ovat erittäin aktiivisia. Angularin yhteisö on myös erittäin suuri ja sillä on paljon kehittäjiä ympäri maailmaa.

Kuitenkin, verrattuna Reactiin ja Vueen, Angularin yhteisö voi olla hieman pienempi ja vähemmän aktiivinen. Yhteisön tuki takaa sen, että kehittäjät saavat apua ongelmiinsa ja löytävät helposti vastauksia kysymyksiinsä. (2.)

Kehittäjien saatavuus on tärkeä kriteeri, sillä tällä hetkellä kehittäjä, ja muut yrityksen kehitystiimin jäsenet ovat tottuneet käyttämään Vue.js:ää verrattuna Reactiin ja Angulariin. Vue on suosittu kehys yrityksen muissa projekteissa.

Työn määrä tarkoittaa sitä, kuinka paljon työtä kehittäjien on tehtävä sovelluksen kehittämiseksi käyttäen tiettyä kehystä. Mitä korkeampi pistemäärä on sitä vähemmän aikaa kuluu dokumentin lukemiseen ja koodin kirjoittamiseen. Asteikko lasketaan näiden tekijöiden yhteismäärän perusteella.

Helppokäyttöisyys, kehittäjien saatavuus ja työn määrä ovat kolme tärkeintä kriteeriä, ja edellä esitettyjen perustelujen ja Vuen arviointiasteikon perusteella päädyimme valitsemaan Vuen projektin selainpuolen kehittämiseen.

Vue.js 3 on uudempi versio kehyksestä, joka sisältää uusia ominaisuuksia ja parannuksia verrattuna Vue.js 2:een. Vue.js 3:n käyttöönotto vaatii muutoksia sovelluksen koodiin, sillä siinä on uusi reaktiivisuusjärjestelmä ja päivitetty syntaksirakenne. (3.) Tämä voi johtaa lyhyellä aikavälillä lisätyöhön ja kustannuksiin sovelluksen kehityksessä. Vaikka Vue.js 3:lla on joitakin etuja, kuten nopeampi suorituskyky ja parempi skaalautuvuus, nämä etuudet eivät ole välttämättä merkittäviä tässä sovelluksessa. (3.) Vue.js 2:lla on edelleen vahva tuki, ja se on todistetusti toimiva kehys meidän monissa erilaisissa sovelluksissa. Lisäksi kehittäjä on kokeneempi Vue.js 2:ssa kuin Vue.js 3:ssa, joten tässä projektissa päätettiin käyttää versiota 2.6.14 selainpuolen rakentamiseen.

Koska Vue on valittu, käyttöliittymän kehittämisessä käytetään BootstrapVueta ja puhdasta CSS-koodia. Palvelinpuoli kommunikoi aina MySQL-relaatiotietokannan kanssa riippumatta, mikä backend-ohjelmistokieli on valittu.

3.1 Vue.js

Vue.js on avoimen lähdekoodin JavaScript-sovelluskehys, joka julkaistiin ensimmäisen kerran helmikuussa 2014. Tällä hetkellä käytössä on kaksi vakaata versiota, Vue 2 ja Vue 3, jotka on suunniteltu nykyaikaisten web-sovellusten rakentamiseen komponenttiarkkitehtuurilla (component architecture). (4.)

Evan You, kehyksen luoja, kertoi, että luotuaan Vuen hän teki sen tarkoituksenaan korjata joitakin puutteita, joita hän oli kokenut Angularissa ja myös Reactissa. Hän kuvaa sitä termillä "progressiivinen ohjelmistokehys", sillä sitä voidaan helposti sovittaa olemassa oleviin projekteihin ja käyttää vain osassa niitä. Vue keskittyy helpottamiseen ja sillä on matala oppimiskäyrä, mikä tarkoittaa, että HTML:n, CSS:n ja Javascriptin perustiedot hallitsevat kehittäjät voivat kokeilla sitä. (5.)

3.2 Vue.js:n ominaisuudet

Komponentit

Vue tarjoaa kehittäjille mahdollisuuden rakentaa sovelluksia helposti yhdistämällä HTML, JavaScript ja CSS samassa tiedostossa SFC-tiedostomuodossa. SFC-tiedostomuodolla kehitettyjen tiedostojen avulla kehittäjät voivat luoda kokonaisia komponentteja yhdessä tiedostossa. Komponenttien uudelleenkäyttö on helppoa ja vähentää tarvetta uudelleenkirjoittaa koodia. (6.)

Yksittäisessä tiedostokomponentissa on seuraavat kolme osaa:

- <template>-osio sisältää komponentin markup-koodin HTML-muodossa, Vuen template auttaa renderöimään HTML:ää dynaamisilla osilla, jotka riippuvat tietomme perusteella.
- script>-osio vie komponentin objektin rakentajan ja koostuu kaikesta JS-logiikasta komponentissa.
- <style>-osio sisältää kaikki komponentin tyylit.

Direktiivit

Direktiivit ovat itse asiassa HTML-attribuutteja, jotka voidaan lisätä HTML-elementteihimme tai komponentteihimme mukautetun toiminnan käynnistämiseksi. Ne alkavat v-merkillä, mikä ilmaisee, että ne ovat erityisiä attribuutteja Vuessa. Sisäänrakennetut direktiivit tarjoavat ominaisuuksia, joita tarvitaan yleisesti sovelluksessa. Esimerkiksi käytetään v-bind, kun asetetaan uudet attribuutit tai ominaisuudet. Käytetään v-html:ää, kun halutaan näyttää raaka-HTML. (6.)

3.3 Vuex

Kun sovellus kasvaa kokonsa ja monimutkaisuutensa myötä, sen tilan hallinta monessa komponentissa voi muodostua hankalaksi. Esimerkiksi kirjautuneen käyttäjän tietojen tai käyttöliittymän tilan esittäminen voi olla vaikea. Ratkaisu tähän ongelmaan on siirtää tila lähimpään yläkomponenttiin, mutta tämä voi johtaa prop drilling -tilanteeseen, jos komponentit eivät ole läheisiä sisaruksia. Prop drilling on tilanne, jossa tieto välitetään yhdestä komponentista läpi useiden toisiinsa liittyvien komponenttien, kunnes saavutaan komponenttiin, jossa tieto tarvitaan. Tämä voidaan tehdä "props ja emits" -mekanismilla, jossa isäkomponentti lähettää tiedon alas lapsikomponentille propsilla tai lapsikoponenti lähettää datat ja tilat takaisin isäkomponentille emitsillä. Mutta monien sisarkomponenttien tai syvän puun kohdalla tämä voi muodostua monimutkaiseksi. Ratkaistakseen tämän suositut kehykset kuten React (Reduxin kanssa) ja Vue (Vuexin kanssa) ovat kehittäneet tilan hallintatyökaluja, jotka tarjoavat yhteisen varaston (store) ja mahdollisuuden päästä käsiksi tilaan minkä tahansa sovelluksen komponentista. Nämä työkalut mahdollistavat myös yhteisen päivityksen tilalle, mikä auttaa estämään virheitä ja epäjohdonmukaisuuksia. (7.)

Kaikki sovelluksemme tiedot ovat yhdessä tietorakenteessa, jota kutsutaan tilaksi ja joka säilytetään varastossa. Varaston käyttöön liittyvät seuraavat periaatteet:

- Sovellus lukee tämän tilan (state) varastosta.
- Tila ei ole koskaan muutettu suoraan varaston ulkopuolella.
- Näkymät lähettävät toimintoja (actions), jotka kuvaavat, mitä tapahtui.
- Toiminnot sitoutuvat mutaatioihin (mutations).
- Mutaatiot muuntavat/muuttavat suoraan varasto-tilaa (store state).
- Kun tila muuttuu, asiaan liittyvät komponentit tai näkymät renderöidään uudelleen. (7.)

3.4 Vue router

Web-kehityksessä reititys viittaa prosessiin, jossa sovellus jaetaan eri osiin URL-mallien (patterns) perusteella. URL muuttuu, kun linkkiä napsautetaan, kuten https://apix.fi:stä https://apix.fi/asiakastuki/:een. Reititys auttaa erottamaan eri osia sovelluksesta, mikä tekee niiden hallinnasta ja ylläpidosta helpompaa. Se myös mahdollistaa turvallisuuden vahvistamisen asettamalla sääntöjä tiettyjen sovellusosien suojelemiseksi. Vaikka sovellusta voi teknisesti kirjoittaa ilman reititystä, se voi muuttua sekavaksi, kun sovellus kasvaa. (8.)

Yksisivuiset web-sovellukset (SPA) ovat verkkosovelluksia, jotka ladataan vain kerran ja käyttävät JavaScriptia dynaamisesti eri sivujen renderöimiseen, mikä parantaa käyttäjäkokemusta. Selaimen ei tarvitse hakea uutta sivua jokaista palvelimeen tehtyä pyyntöä varten (8). Yksisivuisten Vue-sovellusten reititys sisältää kahdenlaista toiminnallisuutta:

- Sovelluksen sijainnin (URL:n) muuttaminen ja määrittäminen, mikä Vue-komponentti renderöidään tietyssä sijainnissa. Tämä tapahtuu käyttämällä Vuen virallista kirjastoa, Vue-routeria.
- Vue-sovellus kartoittaa tiettyjä komponentteja tiettyihin reitteihin ja vaihtaa nykyisen komponentin URL:n perusteella, kun päivitetään URL:n ja renderöimällä oikea komponentti ilman uutta pyyntöä palvelimelle.

3.5 BootstrapVue

BootstrapVue on käärekirjasto (wrapper library), joka yhdistää Vue.js-kirjaston ja suositun CSS-kehyksen Bootstrapin. Se tekee Bootstrapin integroinnista helpompaa Vue.js-sovelluksiin ilman tarvetta käsitellä Bootstrapin raskasta riippuvuutta jQuerystä. Toisin kuin Bootstrap, joka on CSS-kehys, BootstrapVue on käyttöliittymäkomponenttikirjasto. Sitä käytetään sovelluksissa sivujen rakentamiseen ja tarjoamaan interaktiivisia valmiita UI-komponentteja, kuten b-button ja b-alert. BootstrapVuella on monia hyötyjä, kuten valmiit validoinnit käsittelemään lomakkeita. (9.)

3.6 Custom CSS

Tavallisesti BootstrapVue:n sisäänrakennetut komponentit tulevat oletusarvoilla, kuten väri, koko ja tyyli. Nämä suunnittelut ovat joskus tylsiä ja usein toistuvia sovelluksissa, jotka käyttävät tätä frameworkia. Kuva 1 on esimerkkikuva, missä on BootstrapVuen oletuspainikkeet.

Kuva 1. BootstrapVuen oletuspainikkeet.

Kustomoiduilla CSS-tyyleillä voi muokata ulkoasua kehittäjän tarpeiden mukaan samalla hyödyntäen edelleen Bootstrapvuen tehoa ja etuja. Kuva 2 on esimerkkikuva, missä on kustomoidut painikkeet.

Kuva 2. Kustomoidut painikkeet.

Custom-CSS rakennettiin Sassilla käyttäen viimeisintä SCSS-syntaksia. Se on voimakas CSS-esiprosessori, joka laajentaa CSS-kieltä lisäämällä ominaisuuksia, kuten muuttujia, sekoituksia (mixins), funktioita, jotka mahdollistavat CSS:n ylläpidettävyyden ja laajennettavuuden parantamisen (10).

3.7 Go

Go tai Golang on Googlen kehittämä avoimeen lähdekoodiin perustuva ohjelmointikieli. (11.) Go-sovellukset voidaan nopeasti kääntää muotoon, jonka kohdejärjestelmä ymmärtää, verrattuna Java-pohjaisiin ratkaisuihin, joissa on tarpeen runsas konfigurointi ja jotka saattavat vaatia Apache Tomcatin käyttöä.

Go-ohjelmointikieltä kutsutaan usein nimellä Golang sen verkkotunnuksen mukaan, mutta kielen virallinen nimi on Go. Vaikka kieli on ollut olemassa jo jonkin aikaa, sen suosio on lisääntynyt vasta äskettäin pilvipalveluiden ja mikropalveluiden yleistymisen myötä. (11.)

Nykyään kehittäjät joutuvat usein tekemään epämukavan valinnan nopean kehityksen ja suorituksen välillä valitessaan projekteihinsa ohjelmistokielen. C++- ja C-ohjelmistokielet tarjoavat nopeaa suorituskykyä, kun taas Ruby- ja Python-ohjelmistokielet tarjoavat nopeaa kehitystä. Go-suunnittelijat tekivät paljon töitä luodakseen ohjelmistokielen, joka on voimakas ja jossa on ominaisuuksia, jotka tekevät kehityksestä nopeaa ilman tarpeetonta monimutkaisuutta. (12.)

Kehitysnopeus

Go tarjoaa salamannopeita käännöksiä käyttämällä älykästä kääntäjää ja yksinkertaistettuja riippuvuussuhteita jäljitettävyyden ja selkeyden takaamiseksi äskettäin lisätyn Go Modules -ekosysteemin avulla. Go-ohjelmassa kääntäjän tarvitsee katsoa vain kirjastoja, joita tarvitsee suoraan, eikä kulkea kaikkien kirjastojen läpi, jotka sisältyvät koko riippuvuusketjuun, kuten Javassa, C:ssä ja C++:ssa. Tämän menetelmän avulla monet Go-sovellukset on käännetty alle

sekunnissa. Koko Go-lähdekoodipuu on käännetty alle 20 sekunnissa nykyaikaisella laitteistolla. (13.)

Tyyppiturvallisuus

Kun työskentelemme dynaamisen ohjelmointikielen kanssa, olemme saattaneet nähdä kuvan 3 kaltaisen virheilmoituksen.

```
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Kuva 3. Tyyppivirheet. (14.)

Tyyppivirheet tapahtuvat, kun heikosti tyypitetty muuttuja saa arvon, joka on yhteensopimaton sen käytön kanssa muualla ohjelmassa. Ne voivat olla tunnetusti vaikeita debugata, koska emme tiedä, mistä ongelmaisesta arvosta virhe tulee.

Vahvasti staattisesti tyypitetyssä kielessä kuten Go:ssa jokaiselle arvolle on määritetty tyyppi sen luomisen yhteydessä, ja seurantaa hoitaa kääntäjä, mikä tekee tämän tyyppisestä virheestä mahdottoman. (15.) Kuvassa 4 esitetään yhden esimerkin pieni ohjelma, joka tulostaa text-muuttujan arvon näytölle.

```
6
7  func main() {
8          var text string
9          text = 10
10         fmt.Println(text)
11 }
```

Kuva 4. Esimerkkiohjelma. (16.)

Kuvassa 4 ohjelmassa tapahtuu virhe kääntäessä, koska arvo ja tietotyyppi eivät ole samat. Ohjelma näyttää virheen kääntäessä (kuva 5).

```
./prog.go:9:9: cannot use 10 (untyped int constant) as string value in assignment

Go build failed.
```

Kuva 5. Virheilmoitus kääntäessä. (16.)

Struct-tyyppi

Go-ohjelmoinnissa Struct on käyttäjän määrittämä tyyppi, joka tallentaa kokoelman erilaisia kenttiä yhteen kenttään. Struct-tyyppi auttaa organisoimaan ja käsittelemään monimutkaisempia tietorakenteita koodissa. Ne ovat samankaltaisia kuin tietueet (records) tai luokat (classes) muissa ohjelmointikielissä. (17.)

Kuvassa 6 on määritelty CustomerAddress-struct, jolla on ID, osoite, postitoimisto, postinumero ja kaupunki.

```
type CustomerAddress struct {
    ID                  int64 `db:"id_customer_address" json:"id_customer_address"`
    Street1             string `db:"street1" json:"street1,omitempty"`
    PostOffice          string `db:"postal_office" json:"post_office,omitempty"`
    PostCode            string `db:"postal_code" json:"post_code,omitempty"`
    IdCountry           string `db:"id_country" json:"id_country,omitempty"`
}
```

Kuva 6. Structin määrittely.

Kenttiin liitetään myös useita tunnisteita eli tageja, joilla määritellään kentän käyttöä esimerkiksi tietokannan tai JSON-objektien kanssa. Koodissa on käytetty myös lyhennettyjä muotoja "omitempty", jotka tarkoittavat, että jos kentän arvo on nolla tai tyhjä, se jätetään kokonaan pois JSON-objektista.

Lukuisia ohjelmointikieliä on tarjolla. Miksi valitaan Go ? Vaikka työpaikalla pääasiassa käytetäänkin Javaa, on tärkeää huomioida, että Go on myös muiden kehitystiimin jäsen suosittu ohjelmointikieli.

Taulukossa 2 vertaillaan Java- ja Go-kieltä annettujen kriteerien perusteella. Luokittellaan kriteerit asteikolla 1-10. Arvioinnit ovat yleisluonteisia, ja ne perustuvat kehitystiimin kehittäjien henkilökohtaisiin käyttökokemuksiin.

Taulukko 2. Javan ja Go:n vertailu.

Kriteeri	**Java**	**Go**
Helppokäyttöisyys	9	10
Kehittäjien saatavuus	9	9
Työn määrä	9	10

Ensimmäinen kriteeri on helppokäyttöisyys. Go-ohjelmointikielen yksinkertainen syntaksi ja hyvä dokumentaatio tekevät siitä helpon kielen aloitteleville kehittäjille. Lisäksi Go:n asentaminen ja kääntäminen on helpompaa kuin Javan. Go:n asennus ja käyttöönotto ovat suoraviivaisempia kuin Javan, joka vaatii usein monimutkaisempia asennus- ja konfigurointivaiheita. Go:n vakiokirjasto on suhteellisen pieni, ja keskittyy tarjoamaan olennaisia toimintoja. Tämä tarkoittaa sitä, että voidaan usein selviytyä ilman kolmannen osapuolen kirjastoja. (18.)

Toinen kriteeri on kehittäjien saatavuus. Tällä hetkellä kehittäjä ja muut yrityksen kehitystiimin jäsenet hallitsevat sekä Go:n että Javan.

Kolmas kriteeri on työn määrä. Koska Go-ohjelmointikielen yksinkertaisempi rakenne ja selkeämpi koodi tekevät siitä helpomman kirjoittaa ja lukea kuin Java, Go-koodi on yleensä tiiviimpää kuin Javan koodi yksinkertaisemman syntaksin ja vähemmän avainsanojen ansiosta. Nämä johtavat siihen, että myös työmäärä vähenee. Kehitystyö Java-kiellä voi olla tarpeettoman monimutkaista, sillä se vaatii usein paljon koodia ja tiukkaa tyypitystä. (18.)

Helppokäyttöisyys ja työn määrä ovat kaksi tärkeintä kriteeriä, ja edellä esitettyjen perustelujen ja arviointiasteikon perusteella päädyimme valitsemaan Go:n projektin palvelinpuolen kehittämiseen.

3.8 Gorilla Mux

Gorilla Mux on yksi Go:n ekosysteemin suosituimmista reitittimistä kirjastossa. Se on avoimen lähdekoodin kirjasto, joka on rakennettu olemassa olevan net/http-kirjaston päälle. Gorilla Muxin avulla voidaan määritellä monimutkaisia URL-osoiteita ja liittää ne tiettyihin funktioihin tai käsittelijöihin koodissa. Gorilla Mux sisältää myös tuen väliohjelmiston toiminnoille, jotka ovat toimintoja, joita voidaan soveltaa saapuviin pyyntöihin ja muokata pyyntöä tai vastausta jollakin tavalla. Tästä voi olla hyötyä esimerkiksi autentikoinnissa tai lokin kirjoittamisessa. (19.)

3.9 Jmoiron/sqlx

jmoiron/sqlx on suosittu kirjasto Go:n standardikirjaston database/sql-pakettiin, joka on suunniteltu tekemään SQL-tietokantojen kanssa työskentelystä helpompaa. Sqlx tarjoaa joukon hyödyllisiä ominaisuuksia, kuten automaattisen struct-kuvauksen, jonka avulla mapataan tietokannan rivit Go:n structeiksi automaattisesti. (20.)

3.10 Mysql

Kuten useimmissa yrityksessä käytetyissä tietokantoihin liittyvissä sovelluksissa, Mysql:ää käytetään myös tässä projektissa relaatiotietokannan hallintajärjestelmänä.

MySQL on avoimen lähdekoodin relaatiotietokannan hallintajärjestelmä (RDBMS), joka käyttää rakenteista kyselykieltä (SQL) vuorovaikutuksessa tietokantojen kanssa. (21.)

Go-sql-driver/mysql tarjoaa MySQL-ajurin tietokannalle. Sen avulla Go-ohjelmat voivat muodostaa yhteyden MySQL-tietokantoihin ja olla vuorovaikutuksessa niiden kanssa. Se tukee suojattuja yhteyksiä TLS:n avulla, mikä auttaa suojaamaan tietoja kuljetuksen aikana. (22.)

4 Projektin suunnitelma

Tässä luvussa käsitellään projektin arkkitehtuurista ja tarvittavien kirjastojen asentamista. Luku sisältää yksityiskohtaiset kuvaukset projektirakenteesta ja sovellusosien hakemistorakenteesta.

4.1 Arkkitehtuuri

Sovelluksessa käytimme kolmikerrosarkkitehtuuria, jossa on erilliset kerrokset selainpuolille (frontend), palvelinpuolelle (backend) ja tietokannalle (kuva 7). (23.)

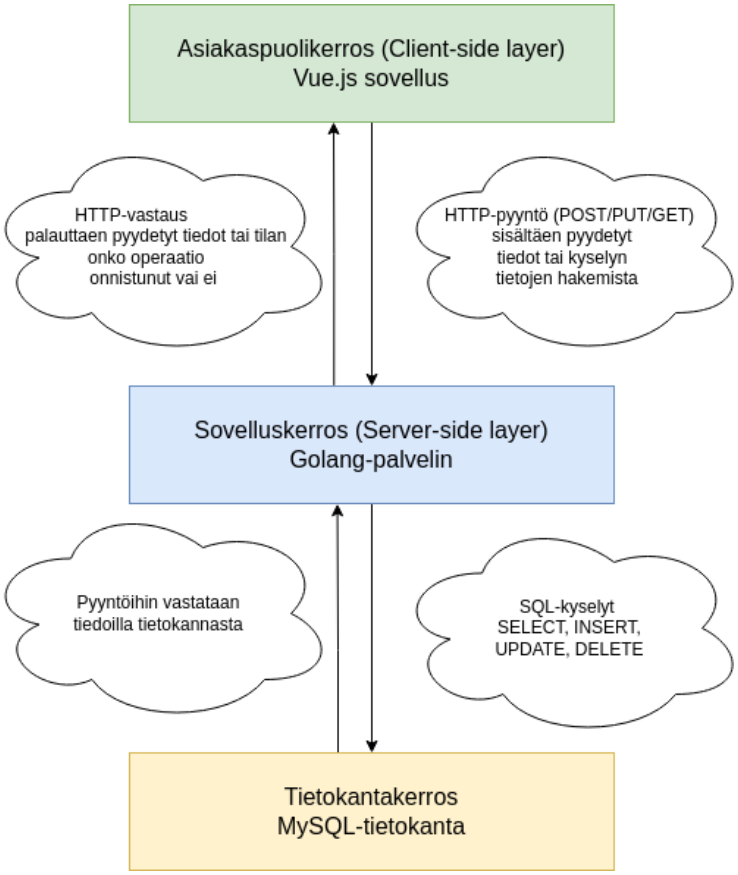
Selainpuolikerros (Vue.js client-side layer) vastaa sovelluksen käyttöliittymästä, joka on nähtävissä käyttäjälle. Käyttöliittymäkerros vastaa myös käyttäjän syötteiden käsittelystä ja lähettää HTTP-pyynnön Go-ohjelmointikielen palvelinpuolelle. Pyyntö voi sisältää tallennettavia tietoja tai kyselyn tietojen hakemista varten. (23.)

Sovelluskerros (Go server-side layer) vastaa sovelluksen logiikasta ja toiminnallisuuksista. Go-ohjelmointikielen palvelin käsittelee HTTP-pyynnön, ja pyynnön tyypistä riippuen se joko tallentaa tietoja MySQL-tietokantaan tai hakee tietoja MySQL-tietokannasta SQL-kyselyillä. (23.)

Tietokanta (MySQL): MySQL-tietokanta tallentaa tiedot ja käsittelee Go-ohjelmointikielen palvelimen lähettämät kyselyt. Kun Go-ohjelmointikielen palvelin lähettää kyselyn, MySQL-tietokanta palauttaa pyydetyt tiedot. (23.)

Kun pyyntö on käsitelty ja tietokannan kanssa on toimittu tarpeen mukaan, Go-ohjelmointikielen palvelin lähettää HTTP-vastauksen takaisin Vue.js:n selainpuolille. Vastaus sisältää pyydetyt tiedot ja tilan, joka kertoo, onnistuiko operaatio. (23.)

Kuva 7 kuvaa sovelluksen arkkitehtuuria.



Kuva 7. Sovelluksen arkkitehtuuri käyttää kolmikerrosarkkitehtuuria.

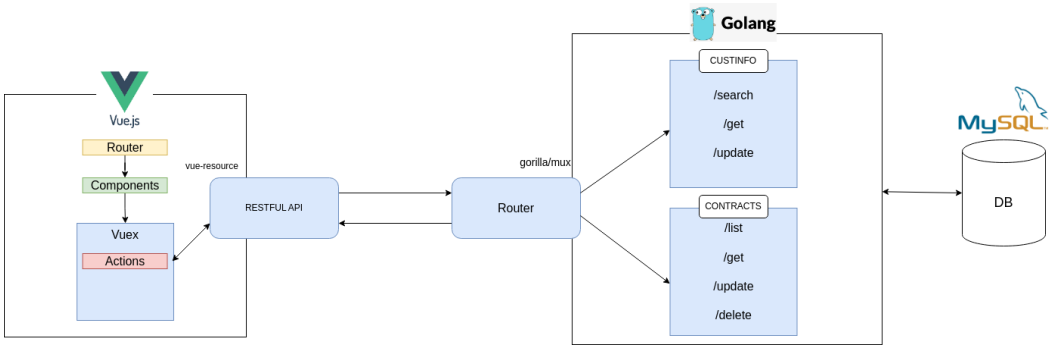
4.2 RESTful API

RESTful API (Representational State Transfer) on ohjelmointirajapinta, joka käyttää HTTP-protokollaa ja sen menetelmiä kuten GET, POST, PUT, DELETE tiedonvaihtoon ja vuorovaikutukseen eri järjestelmien välillä. (24.)

Seuraava luettelo kuvaa RESTful API:n tärkeimmistä ominaisuuksista ja käsitteistä:

- RESTful API perustuu resurssien käsitteeseen, joita ovat esimerkiksi käyttäjät, tuotteet tai tilaukset. Resurssit on yleensä tunnistettavissa URL-osoitteiden avulla. (24.)
- RESTful API käyttää useita HTTP-menetelmiä, jotka määrittävät operaation tyypin resurssin kanssa.
- GET pyytää resurssin tietoja.
- POST luo uuden resurssin.
- PUT päivittää olemassa olevan resurssin.
- DELETE poistaa resurssin.
- RESTful API palauttaa vastaukset yleensä JSON- tai XML. (24.)

Kuva 8 esittää sovelluksen RESTful API -määritykset.



Kuva 8. Sovelluksen RESTful API.

4.3 Kehitysympäristö

Sovelluksen kehitysympäristö toimii Ubuntu 20.04.5 LTS -alustalla. Projektin koko lähdekoodi on kirjoitettu ja editoitu Visual Studio Code -ohjelmalla, VS Studio Code on Microsoftin kehittämä ilmainen, avoimen lähdekoodin koodieditori.

4.4 Selainpuolen riippuvuudet

Seuraavana on luettelo käytetyistä kirjastoista ja riippuvuuksista (dependencies) tuotantoympäristössä:

- Bootstrap-vue on Vue.js-komponentit, jotka on rakennettu Bootstrap CSS:llä.
- Core-js on modulaarinen standardikirjasto JavaScriptille.
- Vue on ydinkirjasto selainpuolien sovelluksien rakentamiseen.
- Vue-clipboard2 on Vue.js-direktiivi, joka lisää leikepöytätoiminnallisuuden komponentteihin.
- Vue-perfect-scrollbar on Vue.js-komponentti mukautettujen vierityspalkkien luomiseen.
- Vue-resource on Vue.js-sovellusten HTTP-asiakaskirjasto, jonka avulla sovellus pystyy suorittamaan asynkronisia HTTP-pyyntöjä ja käsittelemään vastauksia.
- Vue-router tarjoaa selainpuolen reititys ominaisuuksia.
- Vue-select on Vue.js-komponentti pudotusvalikoiden luomiseen.
- Vuex on tilanhallintakirjasto.

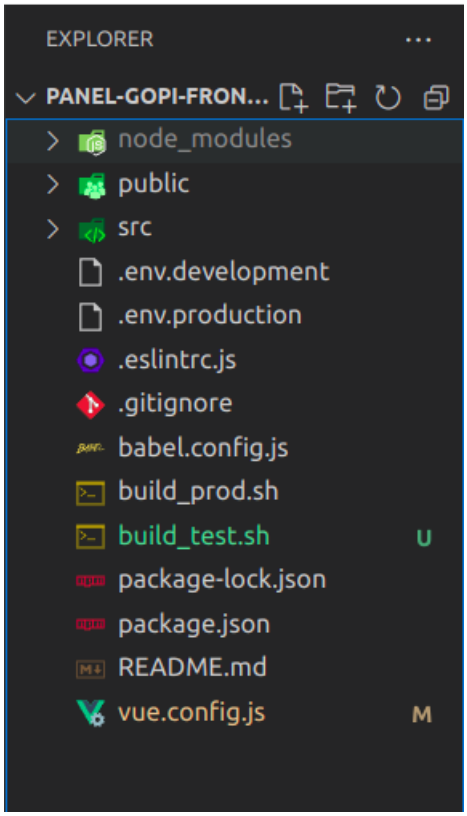
Kuvassa 9 on luettelo yllä mainituista kirjastoista sekä kunkin kirjaston erityinen versio.

```
"dependencies": {
  "bootstrap-vue": "^2.21.2",
  "core-js": "^3.14.0",
  "vue": "^2.6.14",
  "vue-clipboard2": "^0.3.3",
  "vue-perfect-scrollbar": "^0.2.1",
  "vue-resource": "^1.5.3",
  "vue-router": "^3.5.1",
  "vue-select": "^3.11.2",
  "vuex": "^3.6.2"
},
```

Kuva 9. Tuotantoympäristössä käytetty kirjastot ja riippuvuudet.

4.5 Selainpuolen tiedostohierarkia

Ennen projektin aloittamista on erittäin tärkeää määrittää kansiorakenne (kuva 10). Se auttaa meitä järjestämään koodimme siististi ja tekee siitä helpompaa tuleville kehittäjille tehdä muutoksia sovellukseen.

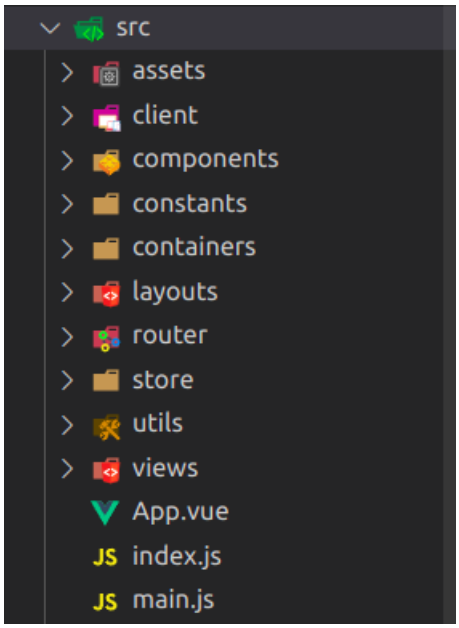


Kuva 10. Vuen projektin rakenne.

Seuraavana on luettelo projektissa käytetyistä kansioista ja tiedostoista sekä niiden tehtävistä:

- /node_modules sisältää kaikki NPM asentamat riippuvuudet.
- /public on erityinen kansio, joka sisältää staattisia resursseja, jotka kopioidaan suoraan build-hakemistoon rakentamisprosessin aikana.
- /src on lähdekoodi kansio, ja se sisältää kaikki tiedostot, joita tarvitaan sovelluksen rakentamiseen ja suorittamiseen.
- .env.development-tiedostoa käytetään kehitysympäristökohtaisten ympäristömuuttujien tallentamiseen. Ympäristömuuttujat ovat avain-arvo-pareja, joita käytetään tallentamaan konfigurointitietoja.
- .env.production-tiedostoa käytetään tuotantoympäristökohtaisten ympäristömuuttujien tallentamiseen. Ympäristömuuttujat ovat avain-arvo-pareja, joita käytetään tallentamaan konfigurointitietoja.
- Build_test.sh on bash-skripti, joka rakentaa Go-sovelluksen testin version.
- Build_prod.sh on bash-skripti, joka rakentaa Go-sovelluksen tuotannon version.

Kansio src on sovelluksen pääkansio, ja se sisältää kaikki tiedostot, joita tarvitaan sovelluksen rakentamiseen ja suorittamiseen (kuva 11).



Kuva 11. Vuen sovelluksen kansiorakenne.

Seuraavana on luettelo kaikista kansioista ja tiedostoista src-kansiossa sekä niiden tehtävistä:

- /assets sisältää staattisia resursseja, kuten kuvia, fontteja.
- /client sisältää metodit, jotka käyttävät Vue-resource HTTP-kirjastoa pyynnön tekemiseen rajapinnalle.
- /components sisältävät dump-komponentteja (dumb components), jotka on useimmiten tehty tarjoamaan näkymiä. Ne saavat tietonsa itsensä ulkopuolelta ja niitä käytetään koko projektissa. Ne huolehtivat siitä, miltä datat näyttävät. Ne eivät ole riippuvaisia muusta sovelluksesta, kuten Vuex-toiminnoista tai Vuex-varastoista. Ne eivät määrittele, miten data ladataan tai mutatoidaan. Niillä on harvoin omaa tilaa jos on. Se on käyttöliittymän tilaa. (25.)
- /constants sisältää kaikki vakioiden tiedostot ja arvot. Kansion tarkoituksena on keskittää vakioiden määrittely ja välttää arvojen kovakoodausta koodin eri osissa.
- /containers sisältävät smart-komponentteja (smart components), jotka voivat sisältää logiikkaa, näkymiä ja muita komponentteja. Ne koskevat sitä, miten datat toimivat. Ne tarjoavat tietoja ja käyttäytymistä dump-komponenteille. (25.)

- /layouts sisältää kaikki sovelluksen layout-komponentit. Ne määrittävät verkkosivujen elementtien järjestely ja asettelu.
- /router sisältää Vue.js-reitittimen määritystiedostot, jotka määrittävät sovelluksen reitit.
- /store sisältää Vuex store -tiedostot, jotka määrittelevät sovelluksen tilanhallinnan.
- /utils on yleinen paikka aputoiminnoille. Niitä käytetään useissa paikoissa.
- /views sisältää sovelluksen päänäkymät.
- App.vue on sovelluksen juurikomponentti.
- Index.js on sovelluksen aloituspiste, ja sitä käytetään Vue.js-instanssin alustamiseen ja sovelluksen mahdollisesti tarvitsemien lisäosien ja kirjastojen määrittämiseen.

4.6 Palvelinpuolen moduulit

Seuraava on luettelo projektissa käytetyistä moduuleista ja niiden tehtävistä:

- github.com/go-sql-driver/mysql v1.6.0 on Go:n MySQL-ajuri, joka mahdollistaa Go-ohjelmien vuorovaikutuksen MySQL-tietokantojen kanssa.
- github.com/gorilla/handlers tarjoaa erilaisia HTTP-väliohjelmistoja Go:n pyyntöjen käsittelyyn.
- github.com/gorilla/mux tarjoaa tehokkaan URL-reitittimen (router) ja -välittäjän (dispatcher), joka sovittaa (matching) HTTP-pyynnöt vastaaviin käsittelijäfunktioihin (handler functions).
- github.com/jmoiron/sqlx on laajennus Go:n SQL-standardikirjastoon, joka tarjoaa joukon apuvälineitä (helpers) SQL-tietokantojen kanssa työskentelyyn.
- github.com/rs/cors tarjoaa CORS-tuen Go HTTP-palvelimille.
- github.com/sirupsen/logrus on Go:n strukturoitu lokikirjasto, jonka avulla kehittäjät voivat tulostaa lokit eri muodoissa, kuten JSON:n.
- github.com/ApixMessagingOy/gopiutils on Apixin omat utils-kirjasto.

Kuvassa 12 on luettelo yllä mainituista moduuleista sekä kunkin moduulin erityinen versio.

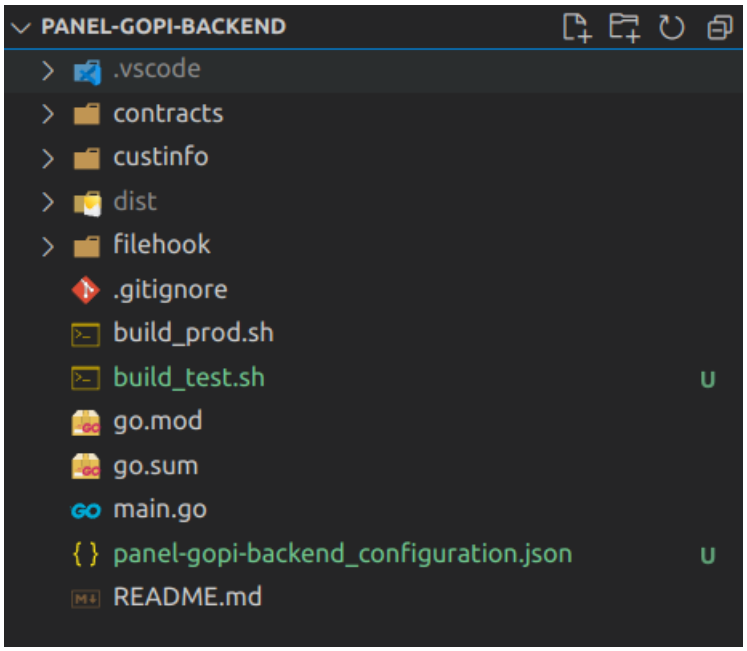
Kuva 12. Go-projektin riippuvuudet.

4.7 Palvelinpuolen tiedostohierarkia

Seuraava on luettelo projektissa käytetyistä kansioista ja tiedostoista sekä niiden tehtävistä:

- /contracts sisältää tiedostoja, jotka käsittelevät pyyntöjä sopimustietojen muokkaamiseksi tai muuttamiseksi käyttöliittymästä.
- /custinfo sisältää tiedostoja, jotka käsittelevät asiakastietojen muokkaus- tai muutospyyntöjä käyttöliittymästä.
- /filehook sisältää tiedosto, joka käsittelee lokin tallentamista.
- Build_prod.sh on bash-skripti, jota käytetään projektin rakentamiseen tuotantoa varten.
- Build_test.sh on bash-skripti, jota käytetään projektin rakentamiseen testausta varten.
- Go.mod sisältää projektin riippuvuudet.
- Go.sum sisältää näiden riippuvuuksien tarkistussummat varmistaakseen, ettei niitä ole muutettu.
- Main.go on ohjelman sisääntulopisteen.
- Panel-gopi-backend_configuration.json sisältää projektin konfiguroinnin tiedot.

Kuva 13 on Go-sovelluksen kansiorakenne.



Kuva 13. Go-sovelluksen kansiorakenne.

5 Toteutus

Sovelluksen suunnitteluvaiheen jälkeen lähdetään rakentamaan sovelluksen näkymiä. Kehitystyö päätetään aloittaa palvelin- ja selainpuolen määrityksestä. Seuraavaksi sovellus luotaisiin osat kerrallaan palvelin- ja selainpuolen kehitystä vuorotellen. Palvelinpuolelle luodaan API-päätepiste ja tämän jälkeen selainpuolelle kehitetään käyttöliittymä, joka mahdollistaa kyseisen päätepisteen kutsumisen ja sen palauttaman datan näyttämisen.

Sovellus on jaettu kahteen päänäkymään. Ensimmäinen näkymä mahdollistaa asiakaslistan selaamisen, ja toinen näkymä näytetään asiakkaan yksityiskohtaiset tiedot.

5.1 Selainpuolen määritykset

Aluksi tiedostossa index.js tuodaan projektin käyttöön tarvittavat CSS-tyylitiedostot, joita ovat bootstrap-vue.css, bootstrap.min.css ja main.scss sekä Vue-router-konfiguraation ja Vuex-konfiguraation tiedostot.

Sen jälkeen asetetaan nämä kirjastot globaalisti Vue-oliolle. Luodaan uusi Vue-instanssi, joka käyttää edellä mainittuja riippuvuuksia ja komponentteja sekä määrittelee renderöintitoiminnon. Sen jälkeen määritellään sovellukselle reitityket (kuva 14).

```
import Vue from "vue";  75.3k (gzipped: 27.3k)

import "bootstrap-vue/dist/bootstrap-vue.css";
import "./assets/css/vendor/bootstrap.min.css";
import "./assets/css/sass/main.scss";

// Router & Store
import router from "./router/";
import store from "./store";

// BootstrapVue
import BootstrapVue from "bootstrap-vue";  531k (gzipped: 142.1k)
Vue.use(BootstrapVue);

// Notification Component
import Notifications from "./components/Notification";
Vue.use(Notifications);

// VueClipboard
import VueClipboard from "vue-clipboard2";  11.1k (gzipped: 3.8k)
Vue.use(VueClipboard);

// VueResource
import VueResource from "vue-resource";
Vue.use(VueResource);

import vSelect from "vue-select";  21.5k (gzipped: 6.4k)
import "vue-select/dist/vue-select.css";
Vue.component("v-select", vSelect);

// Perfect Scrollbar
import vuePerfectScrollbar from "vue-perfect-scrollbar";  25.9k (gzipped: 8.1k)
Vue.component("vue-perfect-scrollbar", vuePerfectScrollbar);

Vue.config.productionTip = false;

export default new Vue({
  router,
  store,
  render: (h) => h(
    <template>
      <div class="h-100">
        <router-view />
      </div>
    </template>
  ),
}).$mount("#app");
```

Kuva 14. Vuen index.js-tiedosto.

Kun käyttäjä tekee pyynnön sovelluksessa, selainpuolen reititys analysoi URL-osoitteen ja määrittää, mitä sivunäkymää tai toimintoa käyttäjä pyytää. Sen jälkeen reititys lataa oikean näkymän tai käynnistää oikean toiminnon. Sovelluksessa on määritelty neljä reittiä käyttäen Vue Routeria (kuva 15).

```
JS index.js M ×
panel-gopi-frontend > src > router > JS index.js > [@] default
1   import Vue from "vue";  75.3k (gzipped: 27.3k)
2   import VueRouter from "vue-router";  29.2k (gzipped: 10k)
3   import { appRoot } from "../constants/config";
4   Vue.use(VueRouter);
5
6   const routes = [
7     {
8       path: "/",
9       component: () => import(/* webpackChunkName: "app" */ "../views/app"),
10      redirect: `${appRoot}/customer`,
11      children: [
12        {
13          path: "customer",
14          component: () =>
15            import(/* webpackChunkName: "customer" */ "../views/app/customer"),
16          redirect: `${appRoot}/customer/search`,
17          children: [
18            {
19              path: "search",
20              component: () =>
21                import(
22                  /* webpackChunkName: "customer" */ "../views/app/customer/Search.vue"
23                ),
24            },
25            {
26              path: "/app/customer/:id",
27              component: () =>
28                import(
29                  /* webpackChunkName: "customer" */ "../views/app/customer/CustomerInfo.vue"
30                ),
31            },
32          ],
33        },
34      ],
35    },
36    {
37      path: "/error",
38      component: () => import(/* webpackChunkName: "error" */ "../views/Error"),
39    },
40    {
41      path: "*",
42      component: () => import(/* webpackChunkName: "error" */ "../views/Error"),
43    },
44  ];
45
46  const router = new VueRouter({
47    linkActiveClass: "active",
48    routes,
49    mode: "history",
50  });
51  export default router;
```

Kuva 15. Vuen reititys.

Juuri-reitti ("/") ohjaa uudelleen "/customer/search"-reitille, joka näyttää hakusivun asiakkaille. "/customer/:id" on dynaaminen reitti, joka näyttää tietyn asiakkaan yksityiskohtaiset tiedot. Se tunnistetaan URL-osoitteen ID-parametrilla. "/error"-reitti näyttää virhesivun, jos navigoinnin aikana ilmenee virheitä, ja "*" -reitti ottaa kiinni kaikki muut reitit ja ohjaa ne virhesivulle.

5.2 Palvelinpuolen määritykset

Aluksi tiedostossa main.go luodaan funktio main(), joka lataa konfigurointin tiedot, luo lokitiedosto ja sitten muodostaa tietokantayhteyden ja luo HTTP-palvelimen (kuva 16).

```
// main loop
func main() {
    log.Infof("load Configurations")
    configuration := loadConfiguration(ConfigurationLocation)

    log.Infof("Add log filehook")
    fileHook, err := filehook.NewLogrusFileHook(configuration.Log.LogFile)
    if err == nil {
        log.AddHook(fileHook)
    }

    log.Infof("init database")
    db := initDataBase(configuration.DataBase)
    if db != nil {
        defer db.Close()
    }

    log.Infof("Init backend")
    initServer(configuration, db)

}
```

Kuva 16. main()-funktio.

Funktio loadConfiguration hakee määritetyt asetukset konfigurointitiedostosta (kuva 17).

```
// TEST
const ConfigurationLocation = "/home/aun/Desktop/PANEELI/panel-gopi-backend/panel-gopi-backend_configuration.json"

// PRODUCTION
// const ConfigurationLocation = "/var/apix/configs/panel-gopi-backend_configuration.json"

func loadConfiguration(configLocation string) utils.Configuration {
    var config utils.Configuration

    data, err := ioutil.ReadFile(configLocation)

    if err != nil {
        panic("Error in reading configuration file " + configLocation + ": " + err.Error())
    }

    err = json.Unmarshal(data, &config)

    if err != nil {
        panic("Error in parsing configuration file " + configLocation + ": " + err.Error())
    }

    return config
}
```

Kuva 17. loadConfiguration()-funktio.

Sitten luodaan lokitiedosto NewLogrusFileHook-fuktiota käyttäen, minkä jälkeen ohjelma kutsuu initDataBase- ja initServer-funktioita.

Funktio initDataBase luo uuden tietokantayhteyden käyttäjän antamien tietojen perusteella. Muodostetaan yhteys tietokantaan SQLX-kirjaston avulla. Jos yhteyttä ei voitu muodostaa, ohjelma lopettaa toimintansa (kuva 18).

```
func dbPing(db *sqlx.DB, seconds int64) error {
    ctx := context.Background()
    ctx, cancel := context.WithTimeout(ctx, time.Duration(seconds)*time.Second)
    defer cancel()

    _, err := db.ExecContext(ctx, "SELECT 1;")

    return err
}

func initDataBase(dbConfiguration utils.DataBaseConfiguration) *sqlx.DB {

    if dbConfiguration.Host == "" {
        log.WithFields(log.Fields{"host": dbConfiguration.Host, "port": dbConfiguration.Port}).Error("database init cancelled, no host defined")
        return nil
    }

    dsn := fmt.Sprintf("%s:%s@tcp(%s:%d)/%s?charset=utf8", dbConfiguration.Username, dbConfiguration.Password, dbConfiguration.Host, dbConfiguration.Port,
    "apix")

    log.WithFields(log.Fields{"host": dbConfiguration.Host, "port": dbConfiguration.Port}).Info("initDataBase / open")
    db, err := sqlx.Open("mysql", dsn)
    if err != nil {
        log.Error("db connection failed: " + err.Error())
        panic("db connection failed:" + err.Error() + "\n")
    }

    log.WithFields(log.Fields{"host": dbConfiguration.Host, "port": dbConfiguration.Port}).Info("initDataBase / set limits")
    db.SetMaxOpenConns(dbConfiguration.MaxOpenConns)
    db.SetMaxIdleConns(dbConfiguration.MaxOpenConns / 2)
    db.SetConnMaxLifetime(time.Duration(15) * time.Second)

    log.WithFields(log.Fields{"host": dbConfiguration.Host, "port": dbConfiguration.Port}).Info("initDataBase / ping database")
    err = dbPing(db, 15)
    if err != nil {
        log.Error("db connection failed: " + err.Error() + "\n")
        panic("db connection failed: " + err.Error())
    }

    db.SetConnMaxLifetime(time.Duration(5) * time.Minute)

    log.WithFields(log.Fields{"host": dbConfiguration.Host, "port": dbConfiguration.Port}).Info("initDataBase / all ok")

    return db
}
```

Kuva 18. initDataBase()-funktio.

Funktio initServer luo HTTP-palvelimen ja luo reitittimen (router), johon määritellään useita polkuja ja niihin liittyviä käsittelijäfunktioita. Sovellus käyttää Gorilla Mux -kirjastoa reitittimen luomiseen. Lopuksi CORS-kirjasto otetaan käyttöön, jotta HTTP-palvelin voi vastaanottaa pyyntöjä muista verkkotunnuksista (kuva 19).

```
func initServer(configuration utils.Configuration, db *sqlx.DB) *mux.Router {
    port := ":" + strconv.FormatUint(uint64(configuration.HTTPServer.Port), 10)

    custinfo := custinfo.Server{Db: db, Servers: configuration.Servers}

    contracts := contracts.Server{Db: db, Servers: configuration.Servers}

    r := mux.NewRouter()

    r.HandleFunc("/custinfo/search", custinfo.SearchCustomerInfoHandler).Methods("POST")
    r.HandleFunc("/custinfo/{IDidentifier:[A-z0-9_-]+}", custinfo.GetCustomerInfoHandler).Methods("GET")
    r.HandleFunc("/custinfo/update/{table}", custinfo.UpdateCustomerInfoHandler).Methods("PUT")

    r.HandleFunc("/contracts/list", contracts.GetContractsListHandler).Methods("GET")
    r.HandleFunc("/contracts/{IdCustomer:[0-9]{1,6}}", contracts.GetCustomerContractsHandler).Methods("GET")
    r.HandleFunc("/contracts/add", contracts.AddCustomerContractHandler).Methods("POST")
    r.HandleFunc("/contracts/update", contracts.UpdateCustomerContractHandler).Methods("PUT")
    r.HandleFunc("/contracts/delete", contracts.DeleteCustomerContractHandler).Methods("PUT")

    c := cors.New(cors.Options{
        AllowedOrigins:   []string{"*"},
        AllowedHeaders:   []string{"*"},
        AllowedMethods:   []string{"GET", "POST", "PUT"},
        AllowCredentials: true,
    })

    handler := c.Handler(r)

    log.Infof("Server running at port %s", port)

    phandler := panicutils.PanicEmailHandler{Handler: handler}

    http.ListenAndServe(port, handlers.RecoveryHandler(handlers.PrintRecoveryStack(true))(phandler))

    return r
}
```

Kuva 19. initServer()-funktio.

Jotta voitaisiin toteuttaa REST API -palvelimen käyttöönotto, joka vastaa asiakastietojen ja sopimusten hallintatarpeisiin, luomme kaksi kansiota nimeltään "custinfo" ja "contracts". Tämän sovelluksen kontekstissa contracts tarkoittaa sitä, että ne ovat Apixin itse määrittelemiä teknisiä sopimuksia.

Kansioon "contracts" luomme muutaman tiedoston, joissa määritetään toiminnot ja tehtävät:

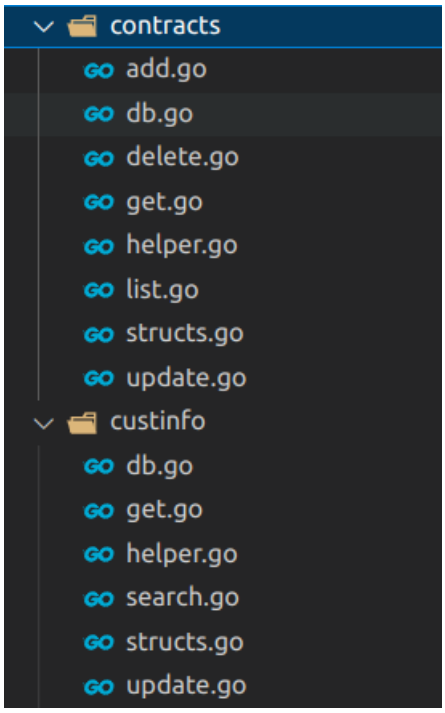
- add.go: Käsittelyfunktio AddCustomerContractHandler käsittelee uuden asiakassopimuksen lisäämisen.

- db.go: Sisältää tietokantakyselyitä tukevat toiminnot.
- delete.go: Käsittelyfunktio DeleteCustomerContractHandler käsittelee olemassa olevan asiakassopimuksen poistaminen.
- get.go: Käsittelyfunktio GetCustomerContractsHandler käsittelee tietyn asiakassopimuksen haun.
- helper.go: Sisältää aputoimintoja, joita käytetään muissa kansiossa "contracts" olevissa tiedostoissa.
- list.go: Käsittelyfunktio GetContractsListHandler käsittelee kaikkien olemassa olevien asiakassopimusten haun.
- structs.go: Määrittää Struct-tyypit, joita käytetään muissa kansiossa "contracts" olevissa tiedostoissa.
- update.go: Käsittelyfunktio UpdateCustomerContractHandler käsittelee olemassa olevan asiakassopimuksen päivittämisen.

Kansioon "custinfo" luomme muutaman tiedoston, joissa määritetään toiminnot ja tehtävät:

- db.go: Sisältää tietokantakyselyitä tukevat toiminnot.
- get.go: Käsittelyfunktio GetCustomerInfoHandler käsittelee tietyn asiakastietojen haun.
- search.go: Käsittelyfunktio SearchCustomerInfoHandler käsittelee asiakkaiden etsiminen tiettyjen kriteerien perusteella.
- helper.go: Sisältää aputoimintoja, joita käytetään muissa kansiossa "custinfo" olevissa tiedostoissa.
- structs.go: Määrittää Struct-tyypit, joita käytetään muissa kansiossa "custinfo" olevissa tiedostoissa.
- update.go: Käsittelyfunktio UpdateCustomerInfoHandler käsittelee tiettyjen asiakastietojen päivittämisen.

Kuva 13 näyttää kaikki luodut kansiot ja tiedostot.



Kuva 20. Go-sovelluksen tiedostot.

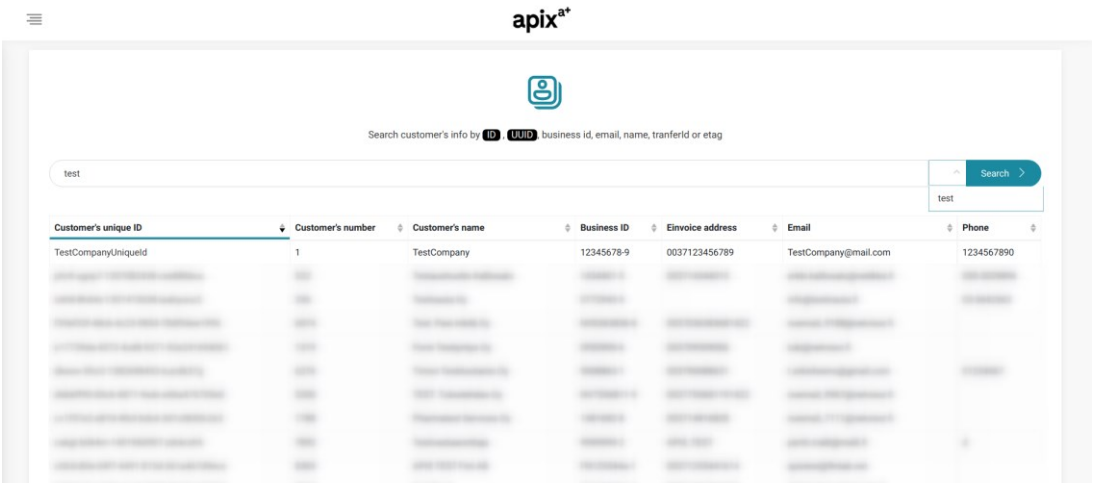
5.3 Kotisivun näkymä

Kotisivun tyylin tulee sisältää lyhyt otsikko , hakupalkki sekä asiakastietojen taulukko.

Otsikko esittelee lyhyesti näkymän ominaisuuksia.

Hakupalkki antaa käyttäjille mahdollisuuden syöttää etsittävät tiedot. Käyttäjät voivat etsiä asiakastietoja nimen, asiakastunnusnumeron, asiakasnumeron, y-tunnuksen, verkkolaskutusosoitteen, sähköpostiosoitteen, puhelinnumeron perusteella. Input-komponentin vieressä on hakuhistoriakenttä, joka tallentaa viimeisimmät hakutulokset.

Asiakastietojen taulukko näyttää asiakastietoja hakusanojen perusteella. Taulukossa on sarakkeita kuten asiakastunnusnumero, asiakasnumero, asiakkaan nimi, y-tunnus, verkkolaskutusosoite, sähköposti ja puhelinnumero. Parantaakseen käyttäjäkokemusta taulukko näyttää tällä hetkellä enintään 10 hakutulosta (kuva 21).



Kuva 21. Kotisivun näkymä.

Hakusivulla käytämme <b-col>-tageja luodaksemme asettelun (kuva 22).

```
<b-col xxs="12">
    <b-card class="mb-4">
    </b-card>
</b-col>
```

Kuva 22. <b-col>.

Seuraavaksi tehdään latausefekti tilamuuttujan isLoading avulla, joka voi olla joko ”true” tai ”false”, joka näyttää tai piilottaa sen (kuva 23).

```
<div v-if="isLoading" class="loading"></div>
```

Kuva 23. Loading-elementti.

Luodaan sivun symboli (kuva 24).

```
<div class="text-center">
  <user-icon />
</div>
```

Kuva 24. Sivun symboli.

Luodaan lyhyt otsikkokuvaus (kuva 25).

```
<div class="pl-3 pr-3 pt-3 pb-0 d-flex flex-column flex-grow-1">
  <p class="text-center mb-4">
    Search customer's info by <span class="hightlight">ID</span> ,
    <span class="hightlight">UUID</span>, business id, email, name, tranferId or etag
  </p>
</div>
```

Kuva 25. Sivun otsikko.

Sen jälkeen luodaan lomake käyttämällä <b-form>-elementtiä. Käyttäjät voivat vuorovaikuttaa sovelluksen kanssa. Lisätään uusi Input-komponentti, johon käyttäjä syöttää etsittävät tiedot, esimerkiksi yrityksen nimen, kuten "testi". Input-komponentilla on attribuutti @change, joka kuuntelee muutoksia. Kaikki muutokset syöttökentässä tallennetaan searchInput-tilamuuttujaan. Käytetään <v-select />-elementtiä, joka luo dropdown-valikon. Dropdown-valikolla käyttäjät voivat valita hakuhistoriansa. Arvo listHistorySearch on hakuhistorialista ja v-model selectedOption on valittu arvo siitä listasta (kuva 26). Lopuksi lisätään hakupainike käyttämällä <b-button>-elementtiä, ja kun käyttäjä napsauttaa painiketta tai painaa ”Enter”, Vue.js kutsuu onSubmit-funktiota, joka suorittaa selainpuolella validoinnin varmistaakseen, että hakutiedot ovat kelvollisia, ja tekee sitten GET-pyynnön palvelimelle JSON-muotoisella {Query: "testi"}.

```
<b-form class="form-container">
  <b-form-group>
    <b-input-group class="search-box mb-4">
      <b-input
        v-model="searchInput"
        :autofocus="true"
        @change="onSubmit"
      />
      <v-select
        v-model="selectedOption"
        :options="listHistorySearch"
      />
      <template v-slot:append>
        <b-button @click="onSubmit" variant="primary">
          <span class="d-inline-block">Search</span>
          <i class="simple-icon-arrow-right ml-2"></i>
        </b-button>
      </template>
    </b-input-group>
  </b-form-group>
</b-form>
```

Kuva 27. listHistorySearch.

Määritetään reiti ”/custinfo/search”-palvelimelle, että Go-ohjelmointikieli pystyy vastaanottamaan haun pyynnön (kuva 28).

```
r.HandleFunc("/custinfo/search", custinfo.SearchCustomerInfoHandler).Methods("POST")
```

Kuva 28. /search-reitti.

Reititys kutsuu käsittelijäfunktiota SearchCustomerInfoHandler, joka tekee yksinkertaisen tarkistuksen varmistaakseen, että tiedot eivät ole tyhjät. Ohjelma kutsuu GetCustomerInfo-funktiota, ja tietokannasta saadut tiedot muunnetaan JSON-muotoon ja palautetaan selainpuolelle. Mikäli suorituksessa tapahtuu virhe, kirjataan virheviestit lokiin ja palautetaan saman tien virheviesti selainpuolelle (kuva 29).

```
func (s *Server) SearchCustomerInfoHandler(w http.ResponseWriter, r *http.Request) {

    responseType := utils.GetResponseType(r)
    reqID := utils.ReqID(r)

    var reqBody SearchString

    err := utils.ParseStructFromRequest(r, &reqBody, false)

    if err != nil {
        utils.HandleErrorToHTTPResponseWriter(responseType, w, err.Error(), http.StatusBadRequest)
        return
    }

    if reqBody.Query == "" {
        logrus.WithFields(logrus.Fields{
            "responseType": responseType,
            "reqID":        reqID,
            "error":        "Query string is empty",
            "function":     "SearchCustomerInfoHandler",
        }).Error("")

        utils.HandleErrorToHTTPResponseWriter(responseType, w, "Query string is empty", http.StatusBadRequest)
        return
    }

    response, err := GetCustomerInfo(s.Db, reqBody.Query)

    if err != nil {

        logrus.WithFields(logrus.Fields{
            "responseType": responseType,
            "reqID":        reqID,
            "error":        err.Error(),
            "function":     "SearchCustomerInfoHandler",
            "context":      "GetCustomerInfo",
        }).Error("")

        utils.HandleErrorToHTTPResponseWriter(responseType, w, err.Error(), http.StatusBadRequest)
        return
    }

    utils.HandleSuccessToHTTPResponseWriter(responseType, w, response)
}
```

Kuva 29. SearchCustomerInfoHandler.

Structs.go-tiedostossa luodaan yhden Struct-tyyppi paulautettava data varten (kuva 30).

```
type CustomerInfo struct {
    UniqueId            string `db:"unique_id" json:"unique_id"`
    IdCustomer          int64  `db:"id_customer" json:"id_customer"`
    CustomerName        string `db:"customer_name" json:"customer_name"`
    VatId               NullString `db:"vat_id" json:"vat_id"`
    CurrentSaldo        string `db:"current_saldo" json:"current_saldo"`
    Email               string `db:"email" json:"email"`
    PhoneNo             string `db:"phone_no" json:"phone_no"`
    ContactPerson       string `db:"contact_person" json:"contact_person"`
    CreatedAt           string `db:"created_at" json:"created_at"`
    EinvoiceAddress     NullString `db:"einvoice_address" json:"einvoice_address"`
    AllowSpam           string `db:"allow_spam" json:"allow_spam"`
    ApixCountry         string `db:"apix_country" json:"apix_country"`
    CustomerType        string `db:"customer_type" json:"customer_type"`
}
```

Kuva 30. CustomerInfo Struct -typpi.

Tiedostossa db.go luodaan GetCustomerInfo-funktio yhdistää sitten tietokantaan tiedon noutamiseksi (kuva 31).

```
func GetCustomerInfo(db dbutils.QueryAbleSqlx, id string) ([]CustomerInfo, error) {

    queryId := "%" + id + "%"

    query := `SELECT
            LIMIT  10;`

    rows, err := db.Query(query, id, queryId, queryId, id)

    if err != nil {
        logrus.WithFields(logrus.Fields{
            "function": "GetCustomerInfo",
            "error":    err.Error(),
        }).Error("Get customer query error")

        return nil, err
    }

    defer rows.Close()

    customers := make([]CustomerInfo, 0)

    err = sqlx.StructScan(rows, &customers)

    if err != nil {
        logrus.WithFields(logrus.Fields{
            "function": "GetCustomerInfo",
            "error":    err.Error(),
        }).Error("Get customers sqlx scan error")
        return nil, err
    }

    return customers, nil

}
```

Kuva 31. GetCustomerInfo.

Selainpuoli lukee tuloksen ja suorittaa tarkastuksen. Jos virheitä esiintyy, käytetään Vue-notifyä näyttämään virheilmoituksen. Jos tarkastus läpäistiin, näytetään se asiakastietojen taulukossa.

Jotta voimme näyttää palvelimen palauttaman tuloksen, luomme taulukon käyttämällä <b-table>-elementtiä. Taulukko näkyy vain, kun siinä on tietoa, joten käytämme v-if="hasData", jossa hasData on totuusarvomuuttuja. <b-table>-elementti tarvitsee kaksi tärkeää parametria: ensimmäinen parametri on ”items”, joka on Javascript taulukko, ja se sisältää asiakasluettelon. Toinen parametri on ”fields”, joka on myös Javascript-taulukko ja se sisältää vastaavat sarakkeen otsikot (kuva 32).

```
<div v-if="hasData">
  <b-table
    responsive
    bordered
    hover
    :items="customers"
    :fields="customerColumns"
  >
    <template #cell(unique_id)="data">
      <router-link :to="`/app/customer/${data.value}`">{{
        data.value
      }}</router-link>
    </template>
    <template #cell(email)="data">
      <span @click="doCopy(data.value)">{{ data.value }}</span>
    </template>
  </b-table>
</div>
```

Kuva 32. Javascript-taulukko.

Asiakastunnusnumero sarakkeessa rivin arvon pitää olla linkki, että käyttäjät pystyvät klikkaamaan linkkiä, joka ohjaa käyttäjät asiakastietosivulle. Tämän toteuttamiseksi käytämme <b-table>:n kanssa seuraavaa syntaksia (kuva 33).

```
<template id="v-step-3" #cell(unique_id)="data">
  <router-link :to="`/app/customer/${data.value}`">{{
    data.value
  }}</router-link>
</template>
```

Kuva 33. <router-link>.

Silloin ”#cell” sisältää sarakkeen unique_id-arvon ja linkin luomiseen käytämme <router-link>.

Vue-table käyttää lajittelujärjestysominaisuutta ”sort”-attribuuteilla, mikä tekee näkymästä intuitiivisemman. Jos käyttäjä haluaa kopioida asiakkaan tietoja, kuten sähköpostia, heidän ei tarvitse valita tietoja, napsauttaa hiiren kakkospainiketta ja kopioida. Vue-clipboard2:n doCopy-fuktion avulla käyttäjän tarvitsee vain napsauttaa kopioitavia tietoja (kuva 34). Tämä lyhentää toimintojen suorittamiseen kuluvaa aikaa ja parantaa merkittävästi käyttäjäkokemusta.

```
<template #cell(email)="data">
    <span @click="doCopy(data.value)">{{ data.value }}</span>
</template>
```

Kuva 34. doCopy.

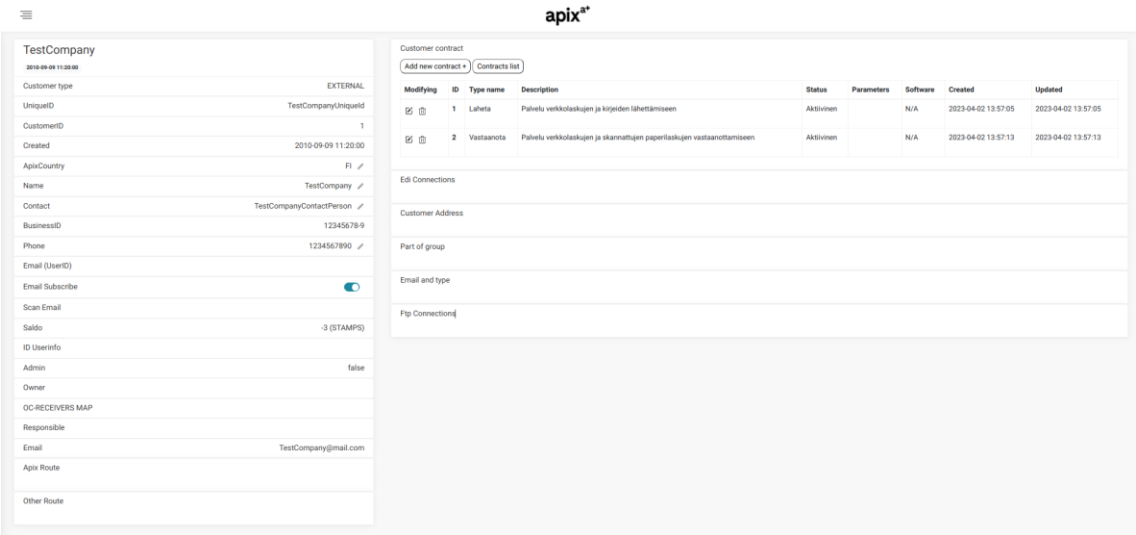
Jos asiakastietoja ei löydy, käytämme <b-alert>-elementtiä, jossa showNotFound on tila ja notFoundMessage on ilmoitusteksti (kuva 35).

```
<b-alert v-if="showNotFound" show variant="danger">
  {{notFoundMessage}}
</b-alert>
```

Kuva 35. notFoundMessage.

5.4 Asiakastietojen näkymä

Asiakastietosivun osalta elementtejä ovat tietotaulukko, kanavat, uuden sopimuksen luontilomake, käytössä olevien sopimusten luettelo, EDI-yhteydet, osoitelistat, sähköpostilistat ja FTP-yhteydet (kuva 36). Kynäkuvakkeella varustetuissa tietokentissä käyttäjät voivat muokata tietoja.



Kuva 36. Asiakastietonäkymä.

Tietotaulukko näyttää asiakkaan yksityiskohtaiset tiedot.

Kanavat ovat luettelo Apixin tai muiden operaattoreiden kautta lähetettäviä tai vastaanotettavia verkkolaskujen kanavia. Verkkolasku tarkoittaa yritykselle tai muulle organisaatiolle lähetettävää sähköistä laskua, joka välittyy vastaanottajalle digitaalisessa ja koneluettavassa muodossa. (26.)

Sopimusluettelo näyttää kaikki asiakkaan käyttämät sopimukset (contracts), esimerkiksi:

- Sopimus nimeltä "Lähetä" antaa asiakkaan lähettää verkkolaskuja.
- Sopimus nimeltä "Receive" antaa asiakkaan vastaanottaa verkkolaskuja.

Uuden sopimuksen luontilomake antaa käyttäjien lisätä uuden sopimuksen asiakkaalle.

Käytössä olevien sopimusten luettelo on lista asiakkaan käyttämistä sopimuksista. Modifying-sarakkeessa käyttäjät voivat suorittaa toimintoja, kuten muokata sopimuksen tietoja tai poistaa sen.

Asiakastietosivu on jaettu kahteen palstaan, joista vasemmanpuoleisen osuus on 33,3 % ja oikeanpuoleisen osuus on 66,6 %. Käytämme <b-row> sekä <b-col xl="4" lg="12"> ja <b-col xl="8" lg="12"> -elementtejä layoutin luomiseen, jotka vastaavat 33,3 %:n ja 66,6 %:n suhteellisia leveyksiä.

Vasemmanpuoleiseen osioon luodaan erillinen komponentti <customer-details>, joka näyttää asiakkaan yleiset tiedot. Käytetään <b-card> ja <b-list-group-item> -elementtejä, joilla luodaan korttiasettelu kahdelle komponentille, Apix Routelle ja Other Routelle. Lisätään sitten <b-table> -elementti ja siirretään tiedot Items-attribuuttiin. Kortti näyttää otsikon ja sisällysluettelon kanssa seuraavalta (kuva 37).

```
<b-card no-body>
  <b-list-group-item>
    <p class="m-0 mr-5">Apix Route</p>
    <b-table
      responsive
      bordered
      hover
      :items="customers.apix_route"
      class="mt-2"
    >
    </b-table>
  </b-list-group-item>
</b-card>
<b-card no-body>
  <b-list-group-item>
    <p class="m-0 mr-5">Other Route</p>
    <b-table
      responsive
      bordered
      hover
      :items="customers.other_route"
      class="mt-2"
    >
    </b-table>
  </b-list-group-item>
</b-card>
```

Kuva 37. Apix Route ja Other Route.

Oikeassa osiossa luodaan myös erillinen komponentti <customer-contract>, joka näyttää luettelon asiakassopimuksista. Edi Connections-, Customer Address-, Part of Group-, Email and Type-, Ftp Connections-komponentit luodaan samalla tavalla kuin Apix Route ja Other Route.

Vue hakee tietoja URL-parametreista ja lähettää pyynnön palvelinpuolelle. Palvelinpuoli hakee tietoja tietokannasta ja palauttaa ne selainpuolelle. Koska erilaisten asiakastietojen näyttäminen vaatii monenlaista tietoa, asiakastietojen haku tietokannasta edellyttää useiden eri taulukoiden yhdistämistä, jotta saadaan halutut tiedot.

CustomerDetails-komponentti koostuu riveistä asiakastietoja, joissa kullakin rivillä on tietokenttä vasemmalla ja vastaava arvo oikealla. Tämäntyyppisen luettelon näyttämiseen käytämme seuraavia elementtejä: <b-card-group>, <b-card> ja <b-list-group>. Koska sovelluksen tavoitteena on mahdollistaa asiakastietojen muokkaaminen, meidän täytyy luoda uudelleenkäytettävä ja tarpeen mukaan muokattava komponentti. Tämän tekemiseksi luomme komponentin nimeltä ListCustomerDetails. Tämä komponentti ottaa parametrinaan (props) otsikon, ja käytämme <slot>-elementtiä lapsikomponenttien tai mukautettujen komponenttien lisäämiseen tarpeen mukaan (kuva 38).

```
<template>
  <b-list-group-item
    class="d-flex justify-content-between align-items-center m-0"
  >
    <p class="m-0 mr-5">
      {{ title }}
    </p>
    <div class="d-flex align-items-center">
      <p class="m-0 ml-5">
        <slot></slot>
      </p>
    </div>
  </b-list-group-item>
</template>

<script>
export default {
  props: ["title"],
};
</script>
```

Kuva 38. ListCustomerDetails-komponentti.

Sitten tuomme yllä luodun komponentin käyttöön ja käytämme sitä nimellä <list-customer-details>.

Tietokentät asetetaan komponenttiin <list-customer-details> käyttämällä otsikkona olevaa ”title”-attribuuttia ja arvon sisältö laitetaan tagin sisälle. Esimerkiksi jos haluamme vain näyttää asiakastiedot, käytämme seuraavaa syntaksia (kuva 39).

```
<list-customer-details title="CustomerID">
  {{ customers.customer_info.id_customer }}
</list-customer-details>
```

Kuva 39. CustomerID.

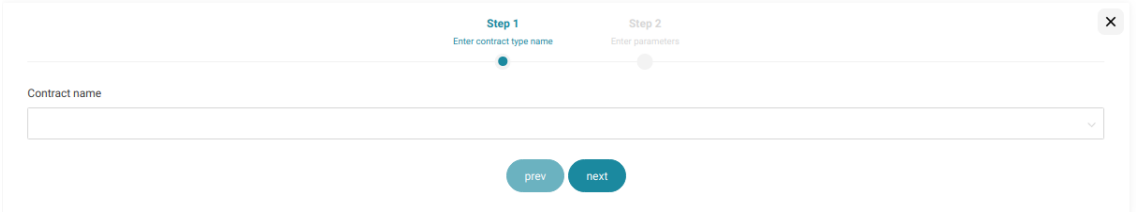
Jos kyseinen tietokenttä on muokattavissa, voimme helposti mukauttaa sitä seuraavasti (kuva 40).

```
<list-customer-details title="Name">
  {{ customers.customer_info.customer_name }}
  <span
    v-if="customers.customer_info.customer_name"
    @click="
      editHandling(
        customers.customer_info.customer_name,
        `Change customer's name`,
        `customer_name`
      )
    "
    v-b-modal.customer-info-edit
    ><edit-icon :fill="`#8a8a8a`" class="edit-button-secondary"
  /></span>
</list-customer-details>
```

Kuva 40. Asiakkaan nimen muokkaus.

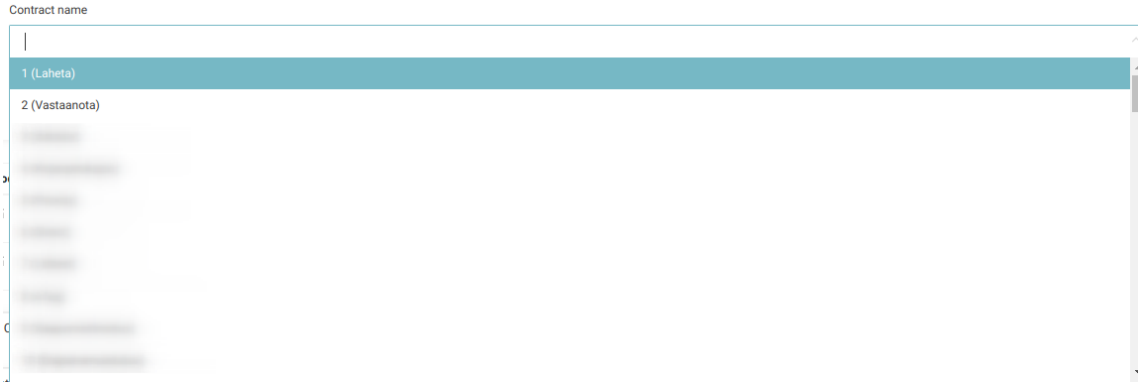
CustomerContract-komponentti on paikka, jossa käyttäjä voi suorittaa toimintoja, kuten lisätä uusia sopimuksia, muokata ja poistaa käytössä olevia sopimuksia.

Lisätäksemme uusia sopimuksia tarvitsemme erillisen komponentin nimeltä NewContractForm. NewContractForm on lomake, johon käyttäjä voi syöttää kaksi arvoa, nimittäin sopimuksen nimen ja sopimuksen parametrin (kuva 41).



Kuva 41. Asiakassopimusluontilomake.

Käytämme <b-form-group label="Contract name"> -elementtiä lomakkeen luomiseen, <v-select>-elementtiä pudotusvalikon luomiseen, joka mahdollistaa olemassa olevien sopimustyyppien valinnan ja <b-form-input>-elementtiä tekstikentän luomiseen (kuva 42).



Kuva 42. Sopimuspudotusvalikko.

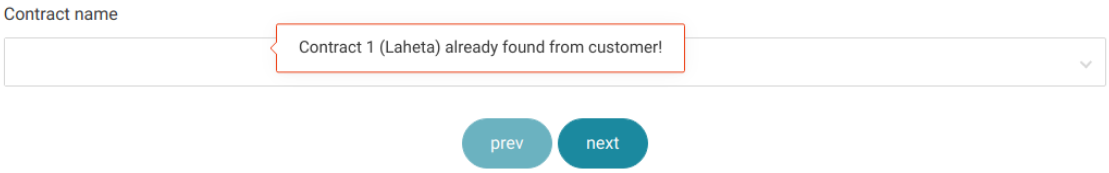
NewContractForm myös validoi käyttäjän syöttämät tiedot, esimerkiksi jos käyttäjä painaa "Jatka"-painiketta mutta sopimus on jo lisätty, validateStep1-funktiota kutsutaan ja ohjelma näyttää virheilmoituksen (kuva 43, 44).

```
validateStep1() {
  this.$v.formStep1.$touch();

  if (!this.formStep1.contractName.value) {
    this.contractNameErrorText = "Contract name is required!";
    this.formStep1.contractName = "";
    return false;
  }
  let arr = this.data.filter(
    (item) =>
      item.id_contract_type.toString() ===
      this.formStep1.contractName.value.toString()
  );
  if (arr.length > 0) {
    this.contractNameErrorText = `Contract ${this.formStep1.contractName.label} already found from customer!`;
    this.formStep1.contractName = "";
    return false;
  }

  return !this.$v.formStep1.$anyError;
}
```

Kuva 43. Sopimuksen validointi.



Kuva 44. Validoinnin virheilmoitus.

Jotta voimme näyttää luettelon käytössä olevista sopimuksista, lisäämme seuraavat komponentit: Lisäämme taulukkoon uuden sarakkeen nimeltä "Modifying", joka sisältää kaksi kuvaketta, eli muokkaus- ja poistokuvakkeet. Kun käyttäjä napsauttaa muokkauskuvaketta, kutsutaan funktiota editHandling. Jos käyttäjä napsauttaa poistokuvaketta, kutsutaan funktiota removalHandling (kuva 45).

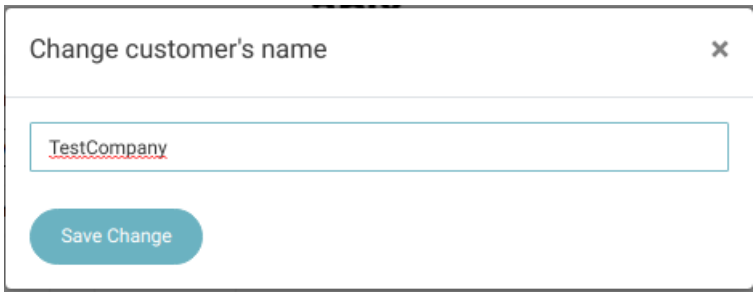
```
<b-table
  responsive
  bordered
  hover
  :items="tableData"
  :fields="contractColumns"
  class="mt-2"
>
  <template #cell(id_customer_contract)="data">
    <p>
      <span
        @click="editHandling(data)"
        v-b-modal.modaledit
        ><edit-icon :color="`#4d4c4c`" class="edit-button"></edit-icon
      ></span>
      <span
        @click="removalHandling(data)"
        v-b-modal.modaldelete
        ><delete-icon class="edit-button"></delete-icon
      ></span>
    </p>
  </template>
</b-table>
```

Kuva 45. editHandling- ja removalHandling-funktiot.

5.5 Modaalinäkymä

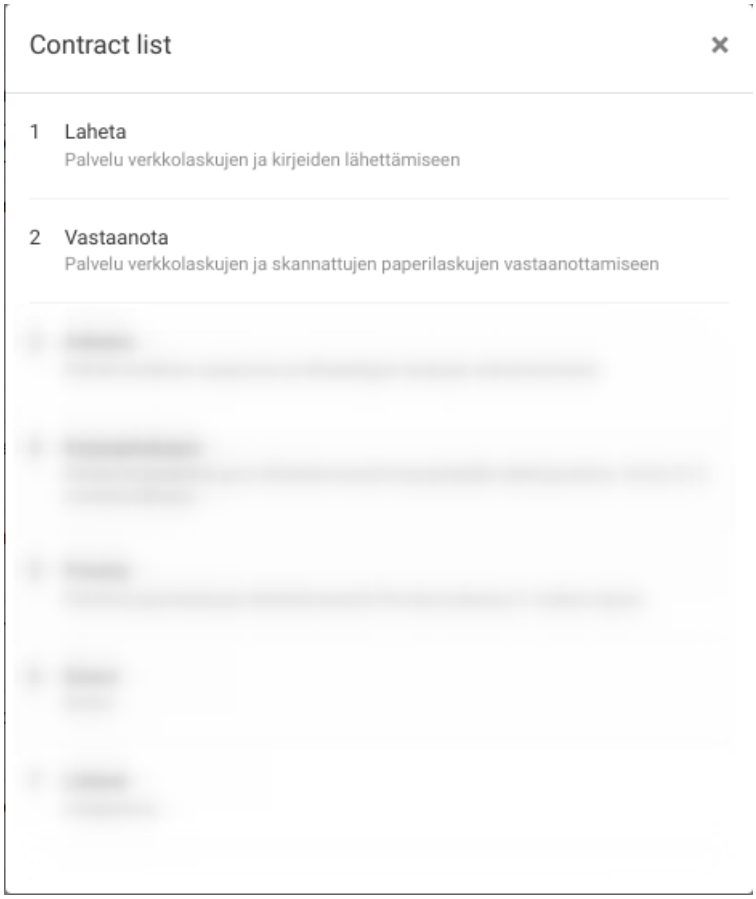
Modal-komponenttia käytetään helpottamaan käyttäjän kanssa vuorovaikutusta muuttamatta sovelluksen perusrakennetta. Sen sijaan että luotaisiin useita erilaisia komponentteja, voimme luoda vain yhden komponentin ja käyttää sitä uudelleen eri tarkoituksiin. Tässä sovelluksessa modaaleja käytetään asiakastietojen muokkaamiseen, asiakassopimusten muokkaamiseen tai olemassa olevien sopimusten poistamiseen.

Kuten edelläkin on jo mainittu, kynäkuvakkeella varustetuissa tietokentissä käyttäjät voivat muokata tietoja (kuva 46). Jotta voimme muokata tietoja, luomme modaalin käyttämällä <b-modal>-elementtiä. Lisäämme input-elementin käyttäjän syöttämien tietojen muokkaamiseen ja lisäämme siihen <button>-elementin, joka tallentaa käyttäjän toiminnan.



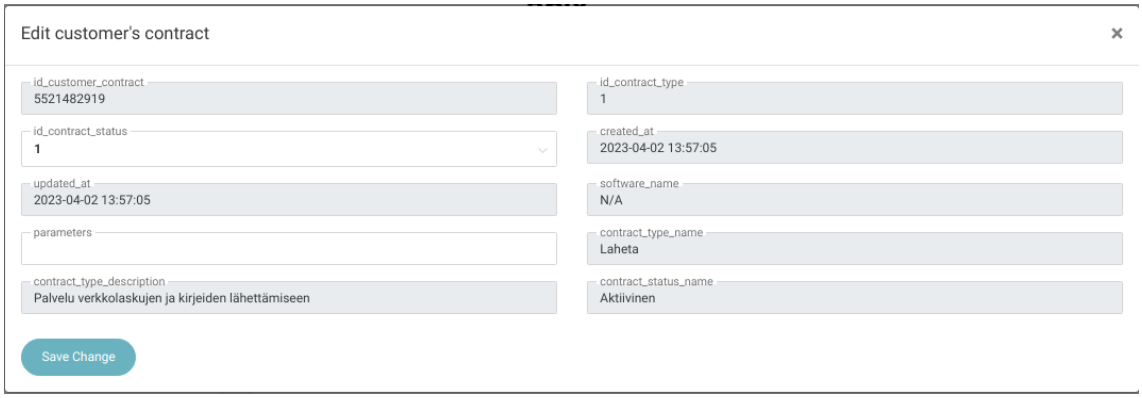
Kuva 46. Asiakastietojen muokkaamiseen.

Jotta voimme näyttää kaikki nykyiset sopimusluettelot (kuva 46), luomme ContractListModal-moduulin käyttäen <b-modal>-elementtiä. Lisäämme lapsielementin <vue-perfect-scrollbar>, jotta vierityspalkki on kauniimpi ja saumattomampi. ContractListModal saa propsina tietueen listan, joka on Javascript-taulukko, ja sitten se renderöidään div-elementteihin.



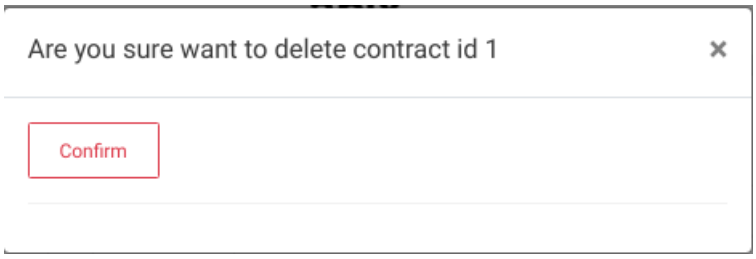
Kuva 47. Sopimusluettelot.

Kun käyttäjä napsauttaa muokkauskuvaketta, avataan ContractEditModal-modaali (kuva 47), joka mahdollistaa käyttäjän muokata sopimuksen tietoja. Käytämme <b-modal>-elementtiä ContractEditModalin luomiseen samalla tavalla kuin muita modaaleja. Käytämme input-elementtiä käyttäjän syöttämien tietojen muokkaamiseen ja lisäämme ”disable"-attribuutin, jos kyseinen tietokenttä ei ole muokattavissa.



Kuva 48. ContractEditModal-modaali.

Jotta voimme poistaa käytössä olevan sopimuksen (kuva 48), kun käyttäjä napsauttaa poistokuvaketta, luomme komponentin käyttämällä <b-modal>-elementtiä ja lisäämme siihen <button>-elementin, joka vahvistaa käyttäjän toiminnan.



Kuva 49. Asiakastietojen poisto.

Tässä osuudessa kerrotaan vaan lyhyesti palvelinpuolen toimintatavan vaiheet:

1. Reititys kuuntelee pyynnöt selainpuolelta.

2. Reititys kutsuu käsittelyfuktiot näistä tiedostosta. Tiedosto get.go jos pyyntö on GET-metodi, tiedosto update.go, jos pyyntö on PUT-metodi, tiedosto add.go jos pyyntö on POST-metodi, ja delete.go, jos pyyntö on DELETE-metodi.

3. Käsittelyfuktiot kutsuvat funktioita tiedostosta db.go.

4. Tietokannasta palautettavat datatietueet vastaavat Structs-tyyppejä, jotka lisätään structs.go-tiedostoon.

5. Tiedostossa db.go-fuktiot hoitavat tietokantakyselyt ja palauttavat datan takaisin käsittelyfuktiolle.

6. Käsittelyfuktio palauttaa virheen tai onnistuneen vastauksen selainpuolelle.

6 Tulokset ja arviointi

Aloitin projektin huolellisesti analysoimalla projektin tavoitteet ja vaatimukset. Asetin selkeän ymmärryksen halutusta toiminnasta ja web-sovelluksen yleisestä tarkoituksesta. Tämä vankka perusta varmistaa, että projekti on selvästi määritelty ja vastaa yrityksen tarpeisiin.

Kun projektin tavoitteet ja vaatimukset oli asetettu, valitsin tehtävään sopivat teknologiat niiden soveltuvuuden perusteella projektiin. Valitsin Go-ohjelmointikielen palvelinpuolen kehitykseen, koska se tarjoaa tehokasta ja luotettavaa suorituskykyä sekä Vuen ja Bootstrapvuen käyttöliittymän kehittämiseen varmistaakseni käyttäjäystävällisen käyttöliittymän ja MySQL:n tietokannan hallintaan.

Kun tarvittavat työkalut oli valittu, suunnittelin käyttöliittymän ja palvelinpuolen komponenttien rakenteen ja organisoinnin. Suunnittelin myös API-päätepisteet ja integroin MySQL-tietokannan asiakastietojen tallennukseen ja hallintaan.

Koko projektin ajan opiskelin ja paransin jatkuvasti taitojani Go-ohjelmointikielessä, Vue.js:ssä, Vuex:ssa ja MySQL:ssä. Etsin aktiivisesti resursseja, kuten dokumentaatioita, verkkokursseja ja kysyin neuvoja työnohjaajalta, jotta projektin eteneminen sujuisi ongelmitta ja pysyisi oikealla tiellä.

Projektin parissa työskenteleminen antaa minulle mahdollisuuden syventää ymmärrystäni Go-ohjelmointikielestä, parantaa luetun ymmärtämistäni ja auttaa minua keräämään arvokasta tietoa sekä saamaan käytännön kokemusta taustakehityksestä. Tämä kokemus on parantanut kykyäni kirjoittaa tehokasta, skaalautuvaa ja ylläpidettävää palvelinpuolen koodia Go-ohjelmointikielellä. Lisäksi olen perehtynyt paremmin kielen parhaisiin käytäntöihin, kirjastoihin ja erityisesti palvelinpuolen kehitykseen suunniteltuihin kehyksiin. Tämä antaa minulle taidot ja tiedot, joita tarvitaan sopeutumiseen ja kehittymiseen dynaamisessa teknologian maailmassa.

Opinnäytetyön tuloksena on toimiva web-sovellus, joka täyttää asetetut tavoitteet ja tuottaa uudelleenkäytettäviä komponentteja ja koodipohjan jatkokehitystä varten. Opinnäytetyön sivutuloksena tekijä on saanut lisää tietoa ja kokemusta käytetyistä tekniikoista sekä oppinut tehokkaampia ohjelmointitapoja.

Vaikka sovellus julkaistiin marraskuussa 2022, on todennäköistä, että lähitulevaisuudessa siihen lisätään uusia ominaisuuksia, jotka palvelevat käyttäjien tarpeita. Opinnäytetyö on ensimmäinen Go-ohjelmointikielellä tekemäni projekti. Projekti oli mielenkiintoinen ja haastava, mutta mielestäni sain sen toteutettua hyvin ja opin paljon uutta. Käyttöliittymän kehittäminen Vuen kanssa ei ole minulle uutta, mutta Go-ohjelmointikielen kanssa olen täysin kokematon. Projektin aikana sain eniten kokemusta Go-ohjelmointikielestä. Taitoni kehittyivät projektin aikana paljon eri osa-alueilla, kuten

tietokantarakenteiden, rajapintojen ja käyttöliittymien suunnittelussa ja toteutuksessa. Henkilökohtaisesti olen tyytyväinen tähän projektiin ja odotan innolla, että pääsen hyödyntämään oppimiani uusia taitoja yrityksen tulevissa projekteissa.

7 Yhteenveto

Opinnäytetyön tavoitteena oli luoda modernin näköinen web-sovellus, websovelluksella pystytään hakemaan, poistamaan, muokkaamaan yrityksen asiakastietoja.

Opinnäytetyön sivutavoitteena oli kehittää tekijän osaamista ja oppia uutta tekniikkaa. Teoriaosuudessa käytiin läpi projektissa käytetyt teknologiat, kuten Go, Vue.js, Vuex, Mysql ja kerrottiin, miten niitä käytettiin projektissa.

Opinnäytetyön tuloksena saatiin toimiva web-sovellus yrityksen asiakkaiden etsimisen varten. Sovellus täyttää asetetut tavoitteet ja tuottaa uudelleenkäytettävä komponentteja ja koodipohjaa jatkokehitykseen. Opinnäytetyön sivutuloksena saatiin lisää tietoa ja kokemusta käytetyistä tekniikoista sekä opittiin tehokkaampia tapoja ohjelmoida.

Lähteet

1 Apix Messaging Oy. 2020. Verkkoaineisto. Apix Messaging Oy. <https://apix.fi>. Luettu 07.01.2023.

2 Pattakos, Aris. 2023. Verkkoaineisto. Angular vs React vs Vue 2023. <https://athemes.com/guides/angular-vs-react-vs-vue/>. Luettu 08.03.2023.

3 Vue docs. 2022. Verkkoaineisto. Vue 3 Migration Guide. <https://v3-migration.vuejs.org/>. Luettu 08.01.2023.

4 You, Evan. 2021. Verkkoaineisto. FIRST WEEK OF LAUNCHING VUE.JS. <https://blog.evanyou.me/2014/02/11/first-week-of-launching-an-oss-project>. Luettu 14.1.2023.

5 Prokop, Paulina. 2019. Verkkoaineisto. Vue.js from scratch - why software developers got so crazy about it. <https://www.merixstudio.com/blog/vuejs-scratch-why-software-developers-got-so-crazy-about-it/>. Luettu 15.1.2023.

6 Djirdeh, Hassan. 2020. Verkkoaineisto. Single File Components. <https://www.newline.co/30-days-of-vue/day-17-single-file-components#single-file-components>. Luettu 21.1.2023.

7 Djirdeh, Hassan. 2020. Verkkoaineisto. Introduction to Vuex. <https://www.newline.co/books/fullstack-vue/intro_to_vuex>. Luettu 22.1.2023.

8 Djirdeh, Hassan. 2020. Verkkoaineisto. Introduction to Vuex. <https://www.newline.co/books/fullstack-vue/intro_to_vuex>. Luettu 22.1.2023.

9 Peter Ekene Eze. 2022. Verkkoaineisto. Getting started with BootstrapVue. <https://blog.logrocket.com/getting-started-bootstrapvue/#why-bootstrapvue>. Luettu 29.1.2023.

10 SASS LANG docs. 2020. Verkkoaineisto. Sass-lang.com. <https://sass-lang.com/>. Luettu 30.1.2023.

11 Go docs. 2023. Verkkoaineisto. Go learn. <https://go.dev/learn/>. Luettu 04.2.2023.

12 Manning. 2020. Verkkoaineisto. Introducing Go. <https://livebook.manning.com/book/go-in-action-second-edition/chapter-1/v-3/14>. Luettu 31.3.2023.

13 Manning. 2020. Verkkoaineisto. Introducing Go. <https://livebook.manning.com/book/go-in-action-second-edition/chapter-1/v-3/20>. Luettu 31.3.2023.

14 Manning. 2020. Verkkoaineisto. Introducing Go. <https://livebook.manning.com/book/go-in-action-second-edition/chapter-1/v-3/22>. Luettu 31.3.2023.

15 Manning. 2020. Verkkoaineisto. Introducing Go. <https://livebook.manning.com/book/go-in-action-second-edition/chapter-1/v-3/26>. Luettu 31.3.2023.

16 Go playground. 2023. Verkkoaineisto. Go playground. <https://go.dev/play/>. Luettu 16.3.2023.

17 Mark McGranaghan and Eli Bendersky. 2023. Verkkoaineisto. Go by Example: Structs. <https://gobyexample.com/structs>. Luettu 12.2.2023.

18 Cordenne Brewster. 2022. Verkkoaineisto. Golang vs. Java: Which Language to Use for Your Next Project. <https://www.trio.dev/blog/golang-vs-java>. Luettu 20.3.2023.

19 Akinyemi, Paul. 2023. Verkkoaineisto. An intro to routing in Go with Gorilla Mux. <https://blog.logrocket.com/routing-go-gorilla-mux/>. Luettu 4.2.2023.

20 Mysql homepage. 2023. Verkkoaineisto. Mysql.com <https://www.mysql.com/>. Luettu 11.2.2023.

21 Moiron, Jason. 2022. Verkkoaineisto. Github branch. <https://github.com/jmoiron/sqlx>. Luettu 12.2.2023.

22 Porada, Phil. 2022. Verkkoaineisto. Go-MySQL-Driver. <https://pkg.go.dev/github.com/go-sql-driver/mysql>. Luettu 20.2.2023.

23 IBM topic. 2023. Verkkoaineisto. What is three-tier architecture. <https://www.ibm.com/topics/three-tier-architecture>. Luettu 21.2.2023.

24 Doyle, Kerry. 2023. Verkkoaineisto. REST (REpresentational State Transfer). <https://www.techtarget.com/searchapparchitecture/definition/REST-REpresentational-State-Transfer>. Luettu 5.3.2023.

25 Abramov, Dan. 2015. Verkkoaineisto. Presentational and Container Components. <https://medium.com/@dan_abramov/smart-and-dumb-components-7ca2f9a7c7d0>. Luettu 8.3.2023.

26 Isolta Oy. 2022. Verkkoaineisto. Opas verkkolaskuttamiseen. <https://www.isolta.fi/opas-verkkolaskutukseen/>. Luettu 5.4.2023.