Antolainausjärjestelmän kehittäminen — Case Profit Software Oy

Tommi Haapa

2022 Laurea

Laurea-ammattikorkeakoulu

Antolainausjärjestelmän kehittäminen – Case Profit Software Oy

Tommi Haapa

Tietojenkäsittely

Opinnäytetyö

Toukokuu 2022

Laurea-ammattikorkeakoulu

Tietojenkäsittely

Tradenomi (AMK)

Tommi Eemeli Haapa

Tiivistelmä

Antolainausjärjestelmän kehittäminen

— Case Profit Software Oy

Vuosi 2022 Sivumäärä 26

Opinnäytetyön tavoitteena oli kehittää osapuolten roolien hallintaominaisuutta Profit Software Oy:n Profit Lainat ja Vakuudet -antolainausjärjestelmään. Työ tehtiin ohjelmointityönä toimeksiantona Profit Softwarelle. Työssä suunniteltiin ja toteutettiin osapuolten roolien hallintatoiminnallisuus valmiina olevaan sovellukseen. Toiminnolle kehitettiin uusi rajapinta ja tietokanta sekä sovelluksen käyttöliittymälle oma paikka, josta rooleja voi hallinnoida. Käyttöliittymän piti olla yhdenmukainen jo olemassa olevan sovelluksen kanssa.

Työn tietoperustassa perehdyttiin teknologioihin ja käytäntöihin, joita käytetään osana ketterää Full Stack -kehitystä. Tietoperustassa keskityttiin osioihin, joita käytettiin toteutuksessa ja mitkä mahdollistivat lopputuloksena syntyneen toiminnallisen lisäominaisuuden. Näitä ovat front-end:issä käytetyt React, Redux, Material-UI-komponenttikirjasto ja TypeScript, sekä back-end-puolen toteutuksessa käytetyt C# -ohjelmointikieli ja Microsoftin ohjelmointiympäristö .NET. Tietokannassa käytettiin SQLServeriä ja RoundHousE -migraatiotyökalua.

Osapuolten roolien hallintaominaisuus saatiin toteutettua, jonka seurauksena roolien hallinta käy kätevästi suoraan käyttöliittymästä, ja manuaalisen työn määrää pystyttiin vähentämään. Sovelluksesta tuli siten myös kokonaisvaltaisempi ja käytettävämpi. Käyttöliittymä saatiin toteutettua yhdenmukaiseksi sovelluksen tyylin kanssa.

Asiasanat: React, Redux, .NET, C#, Scrum, Full-stack, Material-UI

Laurea University of Applied Sciences

Degree Programme in Busines Information Technology

Bachelor's degree

Tommi Eemeli Haapa

Abstract

Development of Loan System

— A Case Study of Profit Software Oy

Year 2022 Pages 26

The purpose of this case study was to develop debtor management feature to Profit Loans and Collaterals -lending management system. Case study was done by programming and as an assignment to Profit Software. Job was to design and implement a feature to manage party roles in already existing application. In terms of functionality, new API, Back-end and database, and of course own location to application, so that different roles could be managed through interface. Accordingly, user-interface needed to be consistent with the existing application.

Several related technologies and practices that are part of agile Full-stack development are presented within the theoretical framework in a more detailed manner. Within the theoretical section, focus was especially on the tools that were used in the assignment, which enabled the successful feature development. These tools, that are commonly used in Front-end include, React, Redux, Material-UI and TypeScript, as well as tools utilized in the Back-end development process such as C# and .NET. In turn, Database development work was made in the SQLServer with the help of RoundHousE -Mirgation tool.

Party role -management feature was developed in order to decrease manual work allowing the usage conveniently through user-interface. As a result of the development process, the usability of the application increased significantly while making it more comprehensive. The development work enabled also more consistent usage of application features through enhanced user-interface.

Keywords: React, Redux, .NET, C#, Scrum, Full-stack, Material-UI

Sisällys

1 Johdanto

Tämän työn tarkoituksena on tehdä osapuolien roolien hallintatoiminnallisuus osaksi jo valmista Profit Lainat ja Vakuudet -antolainajärjestelmää. Profit Lainat ja Vakuudet on laajan toimialaymmärryksen pohjalta rakennettu antolainauksen ja vakuushallinnan elinkaarijärjestelmä. Antolainajärjestelmä koostuu moduuleista, jotka kattavat rahoituksen koko elinkaaren. Näillä kyseisillä moduuleilla hallinnoidaan asiakkaan tarpeita, kattaen lainojen, vakuuksien, sekä luottopäätös- että luottoriskilaskelmat. Uusimpana lisänä on myös tullut tuotehallinta, sekä kirjanpito. Nämä moduulit ovat räätälöitävissä asiakkaan tarpeiden mukaan (Profit a). Lainoilla on aina vähintään yksi osapuoli, joka voi olla esimerkiksi velallinen tai myöntäjä. Joissain tapauksissa antolainajärjestelmän käyttäjät tarvitsevat uusia rooleja osapuolille lainoissa, ennen toteutusta ne on jouduttu lisäämään järjestelmään manuaalisesti suoraan tietokantaan. Toteutuksen tarkoituksena on helpottaa ja nopeuttaa tätä tarvetta. Tavoitteena oli, että rooleja pystytään hallinnoimaan suoraan sovelluksen käyttöliittymältä.

Sovelluksen uusi toiminnallisuus kehitettiin hyödyntäen moderneja teknologioita, ja ketterää ohjelmistokehitystä Scrum -mallilla. Toteutus suoritettiin sprintin aikana sovitussa ajassa. Jo olemassa oleva antolainajärjestelmä on toteutettu React Frameworkilla, ja ohjelmoitu toteutus ohjelmoitiin TypeScriptillä, hyödyntäen React -kirjastoja, kuten Material-UI, Redux ja Redux-form. Ohjelmistoa testattiin Cypressillä ja back-endin yksikkötesteillä. Uusille toiminnoille tehtiin omat yksikkötestit sekä Cypress -testit. Toteutuksen back-end puoli toteutettiin C# -kielellä .NET ympäristössä. Tietokantaa jouduttiin muokkaamaan, jotta haluttu toiminto pystyttiin toteuttamaan. Tietokanta on SQLServerillä ja sitä muokattiin käyttämällä RoundHousE -Migration työkalua.

Jotta toteutus pystyttiin lisäämään tuotannossa olevaan Profitin Lainat ja Vakuudet -antolainajärjestelmään, täytyi toteutukselle tulla hyväksyntä kahdelta muulta ohjelmointitiimin jäseneltä, ja toteutuksen tulee suorittaa yksikkö- ja käyttöliittymätestit onnistuneesti. Hyväksyntää edeltää koodikatselmointi ja toiminnon testaus käyttöliittymältä käsin.

Profit Software Oy on 1992 perustettu suomalainen ohjelmistoyritys, joka tarjoaa ohjelmistoja, konsultointia ja analytiikkaa rahoitus- ja vakuutusalan yrityksille. Profit Softwarella työskenteli 221 ihmistä vuonna 2021, ja sen liikevaihto oli 33 miljoonaa euroa (Vainu).

2 Työn tavoite

Opinnäytetyön tavoite on kehittää olemassa olevaan Profit Lainat ja Vakuudet -antolainausjärjestelmään ominaisuus, joilla lainojen ja vakuuksien osapuolien rooleja pystytään hallinnoimaan käyttöliittymältä. Toteutuksen tulee olla yhdenmukainen muun järjestelmän arkkitehtuurin ja käyttöliittymän puolesta. Käyttäjän lisäämiä rooleja tulee pystyä muokkaamaan, mutta järjestelmän omia ei. Roolit halutaan näyttää taulukossa josta niitä on helppo ja nopea hallita. Järjestelmän omat roolit saavat ID arvot jotka erottuvat käyttäjän lisäämistä.

3 Front-end

Front-end kehittämisellä tarkoitetaan web-kehitystä, joka keskittyy verkkosivuston tai sovelluksen graafisen käyttöliittymän kehittämiseen (Wikipedia b). Front-end voitaisiin myös ymmärtää selainpuolen kehityksenä. Front-end tarkoittaa kaikkea sitä koodia, joka ajetaan selaimessa, eli kaikki mitä käyttäjä näkee, kun sovellusta käytetään (Dagmar 2015).

3.1 React

React on avoimeen lähdekoodiin perustuva Facebookin kehittämä JavaScript kirjasto, jota käytetään rakentamaan käyttöliittymiä (W3schools a). Reactin toiminnallisuus perustuu komponentteihin, käyttöliittymä rakentuu eri komponenteista, ja komponentit ovat JavaScript funktioita (Kuva 1), jotka palauttavat dataa ja käyttöliittymä rakenteen (Jyväskylän yliopiston informaatioteknologian tiedekunta 2022).

```
function Welcome() {
    return <h1>Hello React!</h1>;
}
```

Kuva 1. React komponentti

Reactilla rakennetaan SPA sovelluksia, eli Single Page Application. SPA on sovellus, joka toimii koko elinkaarensa ajan yhdellä sivulla. Käyttäjä lataa käyttöliittymän, joka esittää yhden web-sivun, ja sen jälkeen käyttöliittymää ei tarvitse enää ladata uudestaan, vaan sovelluksen eri komponentit päivittyvät sitä mukaan, kun niitä kutsutaan tai niiden data päivittyy (Tiko Jamk 2020).

3.2 JSX

JSX on syntaksilaajennus, joka on kehitetty JavaScriptille. Sitä suositellaan käytettäväksi yhdessä Reactin kanssa. JSX tuottaa elementtejä, joita voidaan käyttää osana käyttöliittymän rakennusta (React a). Seuraavassa kuvassa on esimerkki, kuinka JSX käytetään (Kuva 2).

```
function formatName(user) {
  return user.firstName + ' ' + user.lastName;
}

const user = {
  firstName: 'Harper',
  lastName: 'Perez'
};

const element = (
  <h1>
    Hello, {formatName(user)}!
  </h1>
);
```

Kuva 2 JSX esimerkki

Kuvassa muodostetaan funktio nimeltä formatName, joka ottaa parametrinä user -objektin. Funktiota kutsutaan <h1> elementin sisällä, näillä parametreillä <h1> elementtiin tulostuisi ”Hello, Harper Perez”. JSX tarkoittaa JavaScript XML, ja se mahdollistaa HTML kirjoittamisen React sovelluksissa (W3schools d).

3.3 Redux

Redux on avoimen lähdekoodin JavaScript-kirjasto, jolla hallitaan sovelluksen tiloja. Sovelluksissa dataa tulee moneen paikkaan, ja Redux tarjoaa ratkaisun hallita tätä dataa järkevästi. Redux tarjoaa keskitetyn varaston sovelluksen tiloille. Näitä tiloja hallitaan sovelluksessa tapahtumilla, joita kutsutaan ”actioneiksi”. (Redux 2021)

3.4 MUI

MUI on React -komponenteille tarkoitettu kirjasto, jonka avulla rakennetaan responsiivisia ja tyylikkäitä käyttöliittymä ratkaisuja, valmiiden MUI komponenttien avulla (Section 2021). Alempana esitellyssä kuvassa on esimerkki, miten MUI kirjaston Button komponenttia käytetään, ja miltä se näyttää koodissa (Kuva 3). Button komponentti tulostaa napin.

```
import * as React from 'react';
import ReactDOM from 'react-dom';
import Button from '@mui/material/Button';

function App() {
  return (
    <Button variant="contained" color="primary">
      Hello World
    </Button>
  );
}
```

Kuva 3 MUI button

Komponentilla on propseja, joilla voidaan määritellä valmiita tyylejä. Buttonin color propsilla määritellään napin väri (MUI). MUI kirjaston komponentit ovat tehty mahdollisimman helppokäyttöisiksi ja omaan tarpeeseen muokattaviksi.

3.5 TypeScript

TypeScript on Microsoftin kehittämä ja ylläpitämä ohjelmointi kieli, joka julkaistiin 2012 (Wikipedia c). TypeScript on vahvasti tyypitetty ohjelmointikieli, joka perustuu JavaScriptiin. TypeScript on kehitetty JavaScriptin päälle helpottamaan isojen sovellusten kehitystyötä ja ylläpitoa. Se lisää tyypityksen JavaScriptiin, joka ennaltaehkäisee virheiden syntymistä, ja edesauttaa virheiden löytymistä (TypeScript 2022). TypeScript osaa kertoa suoraan mistä virheet löytyvät ja mistä ne tulevat. Seuraavassa kuvassa on esimerkki miltä TypeScriptin antama virheilmoitus voisi näyttää.

```
const message = "hello!";

message();

This expression is not callable.
  Type 'String' has no call signatures.
```

Kuva 4 TypeScript virheilmoitus

TypeScript antaa virheilmoituksen, joka kertoo; tätä ei voida kutsua. Määritelty muuttuja message on String tyyppinen, ja sitä yritetään kutsua funktion lailla. Jos tiedosto olisi JavaScriptiä, koodista ei tulisi virheitä. TypeScript ei pelkästään varoita, vaan antaa suuntaa ehdottamalla mahdollisia toimintoja, seuraavassa kuvassa esimerkki ehdotetuista funktioista (Kuva 5).

```
const app = express();

app.get("/", function (req, res) {
  res.sen
                send
});              sendDate
                 sendfile
app.liste        sendFile
                 sendStatus
```

Kuva 5 TypeScript ehdotus

TypeScriptin tyypitys tarkastus siis voi ennaltaehkäistä virheiden syntyä, se osaa kertoa suoraan mitä funktioita voidaan käyttää ja mitä ei. Ylemmässä kuvassa (Kuva 5) on pätkä TypeScript koodia NodeJs:llä toteutetusta rajapinnasta.

Sovelluksissa käytetään paljon objekteja, joihin dataa talletetaan tai missä niitä liikutellaan. Tällöin on erittäin tärkeää, että dataa käsitellään oikein ja oikeilla tavoilla. Alempana on kuva (Kuva 6), joka JavaScriptillä ajettaessa ei antaisi virhettä, mutta palauttaisi ’undefined’. Virhe ei aiheuta virhettä, joka estäisi sovelluksen toiminnan, mutta virheen, jonka seurauksena oikeaa dataa ei tule oikeaan paikkaan.

```
const user = {
  name: "Daniel",
  age: 26,
};

user.location;

Property 'location' does not exist on type '{ name:
string; age: number; }'.
```

Kuva 6 TypeScript virheilmoitus

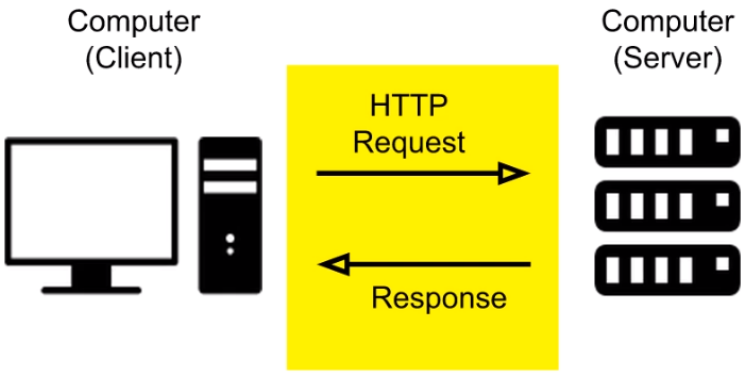
TypeScript kertoo saman tien ilmoituksella, että tällaista tyyppiä ei ole olemassa kyseisessä objektissa. Tämän nähtyään kehittäjän on helppo korjata virhe, ja saada oikea data haluttuun paikkaan.

4 Back-end

Back-end kehityksellä tarkoitetaan kehitystä, joka ei näy käyttäjälle. Back-end koodia ajetaan palvelimella, esimerkiksi serverihuoneessa tai pilvipalvelussa. Back-end puolella tapahtuu sovelluksen lomakkeiden käsittely, kirjautuminen ja salasanojen tarkistaminen ja tietokannan käsittely (Dagmar 2015). Selainpuolelta tulee kutsu back-endille, joka toteuttaa halutun kutsun. Kutsu voi olla, vaikka kirjautumisen todentaminen.

4.1 Rajapinnat

Rajapinta mahdollistaa integraation ohjelmistojen välille. Rajapinnan avulla voidaan tehdä pyyntöjä ohjelmistolle, josta halutaan noutaa, tuoda, viedä tai muokata tietoja (Valjas 2019).



Kuva 7 Rajapinta pyyntö

Rajapintoja voidaan julkaista ainakin kolmella eri tavalla (Wikipedia d):

- Ohjelmointirajapintaa ei julkaista (yksityinen)
- Rajapinta julkaistaan tietyin rajoituksin (yhteistyökumppanit)
- Täysin avoin ja vapaa rajapinta (julkinen)

Rajapinnoille lähetetään HTTP pyyntöjä, yleisemmin pyynnöt tulevat sovelluksen käyttäjältä suoraan käyttöliittymältä, mutta pyyntöjä voi lähettää muualtakin. Ylempänä on kuva jossa on yksinkertainen kommunikaatio palvelimen ja käyttöliittymän välillä. Sovelluksen käyttäjä haluaa kirjautua sisään käyttöliittymältä, joten syöttää kenttiin käyttäjänimen ja salasanan, jotka lähetetään HTTP Requestina eli pyyntönä palvelimen rajapinnalle. Rajapinta kutsuu back-endiä, joka käy tarkastamassa tietokannasta, että löytyykö näillä arvoilla käyttäjää. Kirjautumis tapauksessa vastaus voi olla boolean tyyppinen ”True” tai ”False”. Jos Käyttäjän syöttämät arvot sopivat tietokannassa oleviin käyttäjä arvoihin, palautuu ”True”, jolloin käyttäjä päästetään sisään, muussa tapauksessa palautetaan ”False” ja kirjautuminen estetään käyttöliittymältä.

Yleisimpiä HTTP pyyntöjä ovat (W3schools b):

- GET, joka hakee tietoja
- POST, joka lisää tietoja
- PUT, joka muokkaa tietoja
- DELETE, joka poistaa tietoja

4.2 C#

C# on moderni olio ohjelmointiin tarkoitettu tyypitetty ohjelmointikieli. Sen on kehittänyt Microsoft ja se julkaistiin vuonna 2000. C# ajetaan .NET ympäristössä (Microsoft b). Alempana on esitelty yksinkertainen Hello, World C# syntaksilla (Kuva 8).

```
using System;

class Hello
{
    static void Main()
    {
        Console.WriteLine("Hello, World");
    }
}
```

Kuva 8 C# Syntaksi

C# on vahvasti tyypitettyä, joka tarkoittaa, että funktioiden ottamat parametrit ja niiden palauttamat arvot tulee olla tyypitettyjä. Alempana kuva tyypitetyistä muuttujista (Kuva 9).

```
public class Point
{
    public int X { get; }
    public int Y { get; }

    public Point(int x, int y) => (X, Y) = (x, y);
}
```

Kuva 9 C# Tyypitys

C# on kehittänyt Anders Hejlsberg Microsoftilta, ja kehittäjän mukaan sen tarkoituksena on olla simppeli, moderni ja moneen tarkoitukseen kykenevä olio-ohjelmointi kieli. C# on tarkoitettu kirjoittamaan applikaatioita molempiin sulautettuihin järjestelmiin sekä palvelin järjestelmiin (Wikipedia e).

4.3 .NET

.NET on ilmainen ja avoimen lähdekoodin ohjelmointi alusta, jota voidaan käyttää niin Windowsilla, Linuxilla kuin MacOs:llä. Microsoftin kehittämä ohjelmointi alusta mahdollistaa niin web applikaatioiden tekemisen, pilvi palveluiden rakentamisen, rajapintojen muodostamisen, mobiilisovellusten tekemisen sekä pelien ohjelmoinnin. Yleisimmät kielet .NET ympäristössä ovat C# ja F# (Microsoft a).

5 DevOps

DevOps on toimintamalli, jolla pyritään automatisoimaan sähköisten palveluiden tuotanto. Malli pyrkii automatisoimaan esimerkiksi ohjelmistokehityksen, testaamisen sekä ylläpitämisen. Ohjelmistokehityksessä käytetään ketteriä menetelmiä, sekä jatkuvan toimituksen ja jatkuvan integraation menetelmiä (Wikipedia a).

5.1 Ketterä ohjelmistokehitys

Ketteriä kehitysmenetelmiä on monia, niitä kuitenkin yhdistää ajatus nopeista kehitys sykleistä, joita yleisemmin kutsutaan sprinteiksi. Sprintti on 1–4 viikon mittailen kehitysjakso, jolle scrum tiimi muodostaa tavoitteen ja jakaa tehtävät (Fellowmind). Scrum tiimi muodostuu tuotteen omistajasta, kehittäjätiimistä ja scrum masterista. Scrum master ei ole esimies, vaan tarkkailee ja seuraa työtä, poistaa mahdollisia esteitä kehittämisen edestä tai viestii asioista joka suuntaan. Sprintti alkaa sprintin suunnittelulla, jossa ensin käydään läpi, jäikö viime sprintiltä mitään suorittamatta, jonka jälkeen annetaan uudet tavoitteet ja jaetaan tehtävät. Kehittäjät itse valitsevat mitä kukin tekee, ja vastaa, että tavoite toteutuu. Tiimin tuotokset esitellään sprintin lopuksi yhteisessä katselmoinnissa. Näihin katselmointeihin yleensä osallistuu myös asiakkaita (Fellowmind).

Ketterässä kehityksessä tärkeää on pysyä mukana kehityksessä. Sprintin jokaisena päivänä on Daily eli päivittäispalaveri, joka on lyhyt noin 10-15min kestävä juttu tuokio, jossa käydään läpi mitä kukin on tehnyt edellisenä päivänä, ja mitä aikoo tehdä seuraavaksi. Tämä on myös hyvä paikka kysyä yleisiä asioita, kuten mielipiteitä tietyistä ratkaisuista tai kertoa jos on kohdannut ongelmia kehittäessä.

Tuotteen kehittymisen kannalta on tärkeää, että sprinttiin kuuluu tuotteen kehitysjonon läpikäynti. Kehittäjät keskustelevat ja arvioivat miten uusia kehitettäviä asioita tulisi kehittää, sekä myös antavat työmäärä arvioita (Taskmill 2022).

Sprinttiin kuuluu scrum masterin toteuttama retrospektiivi tai yleisemmin retro, jossa katsotaan ajassa taakseppäin ja keskustellaan edeltäneestä sprintistä, mikä meni hyvin ja mikä

huonosti. Retron tarkoituksena on nostaa kehittäjätiimin motivaatiota ja tuloksia, kuin myös itse prosessin kehittäminen (Scrum). Retron ideana on tuoda esille mahdolliset esteet ja keksiä niihin ratkaisut, mitä ongelmia kohtasimme, miten voisimme ehkäistä tai ratkaista samanlaiset ongelmat.

5.2 Ohjelmistotestaus

Ohjelmisto testaus on ohjelmistokehityksen tärkein tukipilari. Ohjelmistotestauksella pyritään varmistamaan, että sovellus toimii juuri niin kuin sen oletetaan toimivan (Itewiki). Toimiva sovellus tuottaa arvoa käyttäjille, ja huonosti toimiva sovellus karkoittaa käyttäjät. Ohjelmistontestaus yleensä hidastaa kehitystä, mutta se aika joka siihen investoidaan maksaa itsensä takaisin. Mitä aikasemmin virheet huomataan, sitä helpompi niihin on puuttua ja ongelmat korjata. Kattavilla testeillä myös nopeutetaan omalla osalla kehitystä. Jos kehittäjät joutuisivat itse aina käydä sovelluksen toiminnallisuutta läpi jatkokehityksen jälkeen, kuluisi siihen aivan liikaa aikaa.

5.2.1 Yksikkötestaus

Yksikkötestauksessa testataan nimensä mukaisesti yksiköitä, osia koodista tai pieniä ohjelmiston osia. Näillä varmistetaan, että ohjelmiston pienet palaset toimivat oikein. Näitä testejä luodaan aina uusien osien tekemisen jälkeen (Itewiki).

Yksikkötesteillä pidetään yllä metodien toiminnallisuutta, niillä testataan esimerkiksi palauttaako tietyt metodit oikeassa muodossa oikeaa dataa annetuilla parametreillä. Näiden avulla kehittäjän on helppo huomata virheitä nopeasti ja ennen kuin koodia viedään tuotantoon.

Jatkuvassa kehityksessä nämä testit ajetaan pilvessä aina ennen kuin koodi haaroja yhdistetään, eli kehittäjä yhdistää oman kehitys haaransa sovelluksen päähaaraan.

5.2.2 E2E testaus

End to End testaus on niin kutsuttua päästä päähän -testausta. E2E testaus toteutetaan yleensä jonkin kirjaston avulla, yksi suosituimmista E2E kirjastoista on Cypress (Helsingin Yliopisto). E2E testauksen tavoitteena on testata sovelluksen todellisia käyttötarkoituksia. Ne on yleensä robotteja jotka ohjelmoidaan tekemään sitä mitä sovelluksen loppukäyttäjä tulisi sillä tekemään. Alempana on kuva Cypressillä tehdystä yksinkertaisesta testistä (Kuva 7).

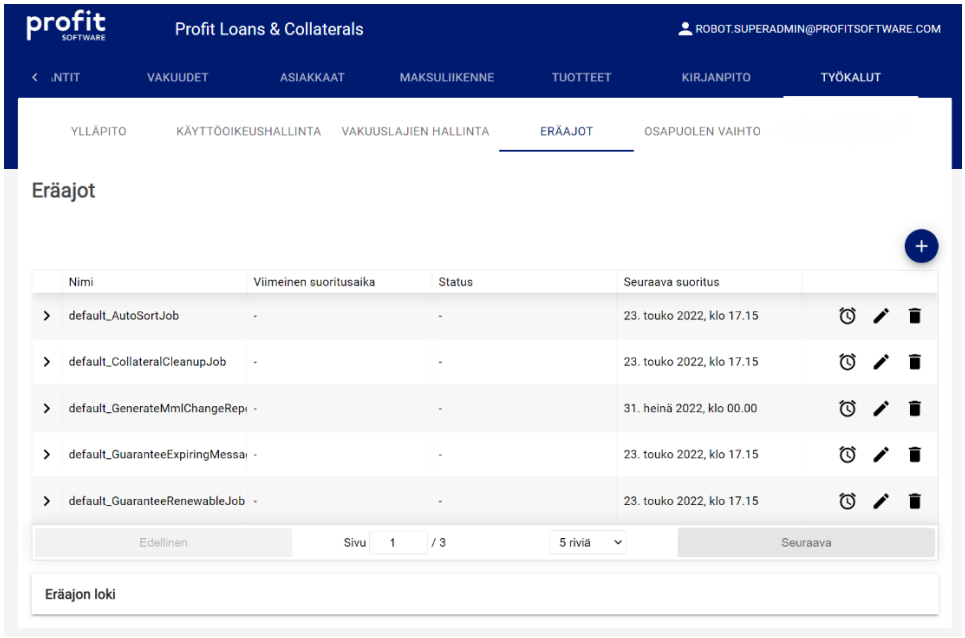
```
describe('My First Test', () => {
  it('clicking "type" navigates to a new url', () => {
    cy.visit('https://example.cypress.io')

    cy.contains('type').click()

    // Should be on a new URL which includes '/commands/actions'
    cy.url().should('include', '/commands/actions')
  })
})
```

Kuva 10 Cypress testi

Ylempänä esitellyssä kuvassa (Kuva 7) on kaksi ohjelmoitua toimintoa. Ensiksi robotti on ohjelmoitu etsimään elementti, joka sisältää sanan ”type”, ja sen jälkeen klikkaamaan sitä. Toiminnon jälkeen suoritetaan tarkastus, urlin tulisi sisältää teksti ”/commands/actions”. Jos urli ei sisällä kyseistä tekstiä, testi epäonnistuu, ja jos sisältää niin päinvastoin, eli menee läpi (Cypress).

6 Toteutuksesta yleisesti

Tässä toteutuksessa kehitetään Profit Lainat ja Vakuudet -antolainajärjestelmään uusi toiminnallisuus, joka mahdollistaa sopimusten osapuolien roolien hallinnan. Sopimuksen osapuolilla voi olla erillaisia rooleja, esim myöntäjä, velallinen tai muu velallinen.



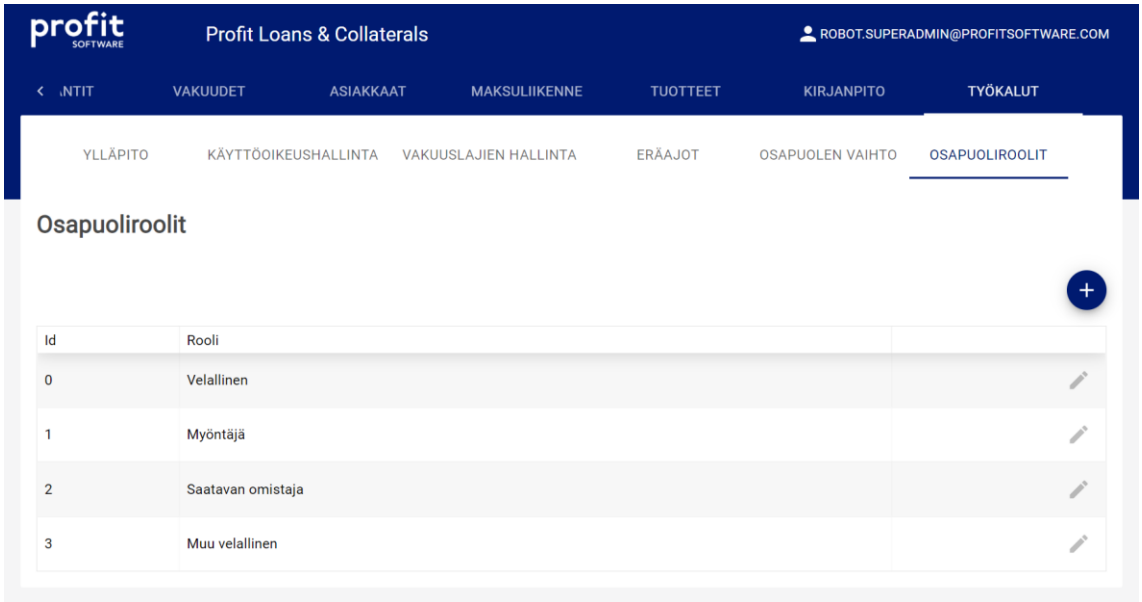
Kuva 11 Työkalut osion valmis osio

Toteutuksen on tarkoitus mahdollistaa näiden roolien hallinta, lisäämisen ja muokkaamisen käyttöliittymän kautta. Toteutuksella on muutama vaatimus;

- Käyttöliittymälle uusi ala-otsikko Työkalut osioon (Kuva 11).
- Ulkoasun tulee olla yhtäläinen muun sovelluksen kanssa (Kuva 11).
- Järjestelmän omia rooleja ei voi poistaa eikä muokata.
- Järjestelmään lisätään uusi rooli Muu velallinen.
- Käyttäjän luomille rooleille on generoitava uniikit ID:t, jotka eivät sekoitu järjestelmän valmiina oleviin rooleihin, käyttäjän lisäämiä rooleja voi muokata ja lisätä, mutta ei poistaa.
- Käyttäjän lisäämät roolit saavat ID arvot alkaen 1000

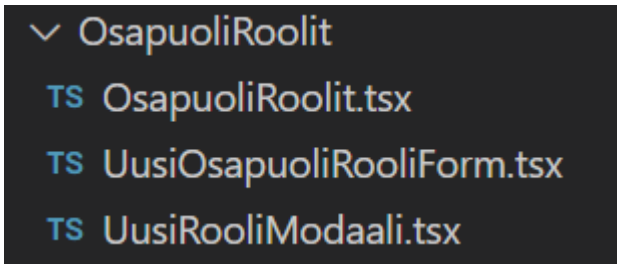
6.1 Käyttöliittymän toteutus

Sovelluksen front-endissä käytetään ohjelmointi kielenä TypeScriptiä, joten tämän osion toteutuksessa käytetään myös sitä. Käyttöliittymän kehitys alkaa lisäämällä oma ala-otsikko haluttuun Työkalut osioon. Työkalut komponentissa on MUI:n komponentti nimeltä Tabs, jonka sisälle on määritelty Tab:eja, jotka näkyvät Työkalut -otsikon alapuolella (Kuva11). Kyseisen Tabs:in sisälle tulee siis lisätä haluttu uusi alaotsikko; OSAPUOLIROOLIT, josta päästään näkymään, jossa pystytään tarkastelemaan olemassa olevia rooleja, lisäämään uusia sekä muokkaamaan lisättyjä (Kuva 27).



Kuva 12 Osapuoliroolit UI

Toteutuksen komponentti rakenne hakemistossa tulee olemaan seuraavanlainen (Kuva 13). Päätason komponentti on OsapuoliRoolit.tsx, joka lisätään Työkalut komponentin Tabs komponentin sisälle omaksi Tabiksi. OsapuoliRoolit komponentin sisälle tulee samanlainen taulukko, kun ylhäällä esitetyssä kuvassa (Kuva 11). Taulukon yläpuolella tulee olla plus -nappula, josta pääsemme modaaliin mistä pystymme lisätä uuden roolin.



Kuva 13 Osapuoliroolien hakemisto rakenne

OsapuoliRooli taulukkoon tarvitaan arvoja, joita saamme reduxin avulla. Reduxilla lähetämme pyynnön sovelluksen back-endille (Kuva 14).

```
export function getAllPartyRoles() {
  return async (dispatch: RootDispatch) => {
    dispatch(load("partyRoles"));

    const url = `${agreementsApiUrl}/api/OsapuoliRooli`;

    try {
      const res = (await request.getResponseJson<Roolit[]>(url)) ?? [];
      const data = fromJS(res);
      dispatch(loadSucces("partyRoles", data));
    } catch (error) {
      await handleFetchError(dispatch, error, "Failed to load party roles", () =>
        loadFail("partyRoles", "Unable to load party roles"));
    }
  };
}
```

Kuva 14 Redux funktio

Redux on React kirjasto, jonka avulla voimme hallita sovelluksen tiloja ja dataa. Reduxin avulla teemme kutsun rajapinnalle joka hakee tietokannasta olemassa olevat roolit. Pyynnön vastaus talletetaan reduxin omaan varastoon, jota kutsutaan stateksi, josta voimme tulokset hakea myöhemmin (Kuva 14). Funktio palauttaa listan jossa on kaikki järjestelmän roolit.

OsapuoliRoolit komponentin sisälle tehdään Reactin oma useEffect hookki, jonka sisällä kutsutaan Reduxiin tehtyä getAllPartRoles funktiota. Reduxista saa helpoiten dataa niin kutsutuilla selectoreilla. Ne ovat funktioita, jotka palauttavat Reduxin omasta varastosta tiettyä dataa. Toteutin Redux selectorin (Kuva 15), jonka avulla tallensin selectorin palauttaman datan muuttujaan.

```
export const partyRolesSelector = (state: RootState) =>
  (state.get("partyRequestData")?.get("partyRoles")?.get("data") as TypedList<Roolit>) ??
  (List() as TypedList<Roolit>);
```

Kuva 15 Redux selectori

Toteutuksen tulee olla yhtäläinen muun sovelluksen kanssa, joten selectorilla haettu data pitää ohjata samanlaiseen tauluun, kun yllä esiteltiin (Kuva 11). Sovelluksessa on käytetty React-table kirjaston taulukko komponenttia nimeltä ReactTableWrapper, jolle pitää antaa parametrinä tarvittavat kolumnit, sekä haluttu data. Roolilla on kaksi arvoa, Selite joka on roolin nimi, sekä SopimusOsapuoliRooli_ID joka on roolin id arvo. Täten tarvitaan kolme kolumnia taulukolle, ID, Rooli sekä yksi ylimääräinen johon tulee iconi mistä pystymme muokkaamaan rooleja. Nämä kolumnit annetaan ReactTableWrapperille, joka muodostaa yhtäläisen taulukon edellä mainituista kolumneista ja datasta.

Käyttöliittymälle tarvitaan myös mahdollisuus lisätä uusia rooleja. Tähän tarkoitukseen tein OsapuoliRooli komponenttiin plus -napin, jota painamalla avautuu modaali, jossa pystymme lisäämään uuden roolin (Kuva 16).

Kuva 16 Uusi rooli modaali

Loin modaalin käyttämällä MUI:n tarjoamaa valmista Dialog komponenttia. Modaalin sisällä on yksinkertainen formi UusiOsapuoliRooliForm (Kuva 16), joka TALLENNA nappia painamalla ottaa formin parametrit ja lähettää ne Reduxin funktion avulla back-endille, joka hoitaa tallentamisen tietokantaan.

Toteutuksen vaatimuksissa määriteltiin vielä, että käyttäjän lisäämiä rooleja tulee pystyä muokkaamaan. Taulukon oikeaan reunaan tehtiin kynä iconi, jota painamalla avautuu sama modaali mikä ylempänä esiteltiin, mutta siinä on valmiina valittu rooli (Kuva 11). Kun muokkaus on suoritettu, muokattu rooli lähetetään Reduxin avulla back-endille, joka hoitaa tietojen päivityksen. Taulukon riveiltä disabloitiin mahdollisuus painaa muokkaus nappulaa jos rivin roolin ID arvo on alle 1000.

6.2 Rajapinta

Sovelluksen back-end on toteutettu .NET 6 CORE:lla käyttämällä C# ohjelmointi kieltä. Toteutuksen vaatimuksissa oli, että rooleja voi lisätä, sekä muokata, mutta ei poistaa. Tämän pohjalta tulee sovelluksen back-end:ille tehtävä ainakin rajapinnat GET, PUT sekä POST. Aluksi tulee suunnitella mitä rajapintojen tulisi tehdä, eli minkälaisia metodeita minkäkin takana tulisi olla. Ensimmäinen mitä tarvitaan, on GET rajapinta joka palauttaa järjestelmän kaikki roolit. Tämä onnistui luomalla GET rajapinta (Kuva 15). Rajapinnalle määriteltiin mitä sen pitää palauttaa, rajapinta ei ota vastaan mitään parametreja. Rajapinta palauttaa listan rooleja. Rooli koostuu string kentästä Selite, ja int kentästä SopimusOsapuoliRooli_ID.

```
[HttpGet]
[Authorize(Util.Permissions.LoansRead)]
[ProducesResponseType(typeof(RoolitDto), StatusCodes.Status200OK)]
1 reference | Jussi Kauhanen, 60 days ago | 2 authors, 3 changes | 99 work items | 0 requests, - live | 0 exceptions, - live
public async Task<ActionResult<List<RoolitDto>>> GetPartyRolesAsync()
{
    return Ok(await _osapuoliRooliManager.GetAsync());
}
```

Kuva 17 GET rajapinta

Rajapinta kutsuu tiettyä metodia, joka hoitaa halutun toiminnon. Osapuoli rooleille tehtiin oma manageri, johon luotiin eri metodeita joita rajapinta pystyy käyttämään. Kun käyttöliittymältä tulee kutsu GET rajapintaan, odotetaan managerin GetSync metodia, joka palauttaa tietokannasta kaikki roolit mitä löytyy.

```
5 references | Jussi Kauhanen, 60 days ago | 2 authors, 2 changes | 99 work items | 0 exceptions, - live
public async Task<List<RoolitDto>> GetAsync()
{
    var roolit = await db.SopimusOsapuoliRooli.OrderBy(x => x.SopimusOsapuoliRooli_ID).ToListAsync();

    return _mapper.Map<List<RoolitDto>>(roolit);

}
```

Kuva 18 GetAsync metodi

Managerin metodi GetSync hakee tietokannasta taulusta SopimusOsapuoliRooli kaikki mitä löytyy, sekä vielä järjestää ne SopimusOsapuoliRooli_ID arvon mukaan järjestykseen OrderBy metodin avulla (Kuva 18). Metodi on asyncroninen, joten se odottaa kunnes pyyntö on toteutettu. Kun toiminto on toteutettu, palautetaan listassa kaikki roolit mitä löydettiin.

Nyt back-endiltä voidaan pyytää käyttöliittymälle kaikki roolit, jota järjestelmästä löytyy. Seuraavaksi täytyy toteuttaa toiminnallisuus jolla voimme lisätä järjestelmään uusia rooleja. Aloitetaan tekemällä POST rajapinta, ja tälle metodi CreateAsync, joka lisää tietokantaan halutun roolin (Kuva 19). POST rajapinta on kuin GET rajapinta (Kuva 17), mutta sille tulee antaa parametrinä uusi rooli. GET rajapinnasta poiketen POST rajapinta palauttaa luodun roolin, tai virheen tullessa kertoo virheestä.

```
5 references | Jussi Kauhanen, 60 days ago | 2 authors, 2 changes | 99 work items | 0 exceptions, - live
public async Task<RoolitDto> CreateAsync(RoolitDto rooliDto)
{
    var rooli = _mapper.Map<SopimusOsapuoliRooli>(rooliDto);

    try
    {
        db.SopimusOsapuoliRooli.Add(rooli);
        await db.SaveChangesAsync();
    }
    catch (ConflictException ex)
    {
        throw new ConflictException($"Role name {rooliDto.Selite} already exists", ex);
    }

    return await GetByIdAsync(rooli.SopimusOsapuoliRooli_ID);

}
```

Kuva 19 Uuden roolin lisäys järjestelmään

CreateAsync metodi ottaa parametrinä käyttäjän haluaman uuden roolin. Käyttäjän syöttämä rooli käännetään tietokanta muotoon Map funktion avulla (Kuva 19). Mäpätty uusi rooli yritetään lisätä tietokantaan try-catch tyylillä. Ensiksi yritetään lisätä tietokantaan, ja jos ei onnistuta, otetaan virhe kiinni ja palautetaan käyttöliittymälle ilmoitus virheestä. Tässä metodissa halutaan napata ConflictException, joka on C# oma virheilmoitus, jos tietokannasta palautuu tieto, että vastaava tieto löytyy jo (Kuva 19).

```
7 references | Jussi Kauhanen, 60 days ago | 2 authors, 2 changes | 99 work items | 0 exceptions, - live
public async Task<RoolitDto> UpdateAsync(int id, RoolitDto rooliDto)
{
    try
    {
        var rooli = db.SopimusOsapuoliRooli.SingleOrDefault(x => x.SopimusOsapuoliRooli_ID == id);

        if (rooli == null)
        {
            _logger.LogInformation("No partyrole found with Id: {id}", id);
            throw new EntityNotFoundException($"No partyrole found with Id: {id}");
        }

        _mapper.Map(rooliDto, rooli);

        await db.SaveChangesAsync();
    }
    catch (ConflictException ex)
    {
        throw new ConflictException($"Role name {rooliDto.Selite} already exists", ex);
    }

    return await GetByIdAsync(id);
}
```

Kuva 20 Muokkaa roolia -metodi

Vielä puuttuu toiminnallisuus jolla pystytään muokata jo olemassa olevaa käyttäjän luomaa roolia. Luodaan rajapinta PUT, joka saa käyttöliittymältä muokkauspyynnön mukana

muokattavan roolin ID arvon, sekä uuden muokatun roolin. Muokattavan roolin ID:tä tarvitaan managerin metodille, jotta pystytään muokata tietokannassa oikeaa roolia. Manageriin luodaan metodi, joka ottaa vastaan ID arvon, sekä uuden muokatun roolin (Kuva 20). Metodi käyttää aikaisemmin mainittua try-catch metodia. Try:n sisällä ensiksi haetaan tietokannasta muokattava rooli ja tallennetaan se muuttujaan. Muuttujan alapuolella on ehtolause, jos kyseistä roolia ei löydy tietokannasta. Palautetaan virhe joka kertoo, että kyseisellä ID arvolla ei löydy roolia. Jos rooli löytyy, päivitetään rooliksi käyttäjän syöttämä uusi rooli Map funktion avulla (Kuva 19). Käyttöliittymälle palautetaan muokattu rooli, tai virheen tullessa virhe.

6.3 Tietokanta

Tietokantana toimii Microsoftin SQLServer 2019 versio. Tietokannan suunnittelun aloitin hahmottelemalla minkälaisia tietokanta muutoksia tulisi tehdä. Järjestelmässä on aloitus hetkellä kolme valmista roolia, Velallinen, Myöntäjä ja Saatavan omistaja. Järjestelmään haluttiin lisätä uusi rooli Muu velallinen. Nykyinen SopimusOsapuoliRooli taulu on erittäin yksinkertainen (Kuva 21), mutta ongelmana on, että jos uusi rooli lisätään tauluun, tulee sille SopimusOsapuoliRooli_ID 4. Käyttäjän lisäämille rooleille haluttiin generoida uniikit ID:t, jotta ne eivät sekotu järjestelmän omien roolien kanssa.

	SopimusOsapuoliRooli_ID	Selite
1	3	Muu velallinen
2	1	Myöntäjä
3	2	Saatavan omistaja

Kuva 21 SopimusOsapuoliRooli -taulu

Jos halutaan generoida uniikit ID:t lisätyille rooleille, tarvitsee tehdä muutoksia tietokanta tauluun. Taulun nykyinen ID: kenttä ei omista IDENTITY ominaisuutta, joka generoi automaattisesti uuden ID:n halutussa järjestyksessä. Aloitin toteutuksen poistamalla olemassa olevan tietokanta taulun, jotta voin toteuttaa sen toisella tavalla. Taulu poistetaan tietokannasta SQL skriptin avulla (Kuva 22). Ensiksi poistetaan taulun viiteavaimet muihin tauluihin jossa rooleja on käytössä DROP CONSTRAINT komennolla. Kun viiteavaimet on poistettu, voidaan taulu poistaa DROP TABLE komennolla.

```
ALTER TABLE [dbo].[TuoteOsapuoli]
DROP CONSTRAINT Fk_TuoteOsapuoli_SopimusOsapuoliRooli
GO

ALTER TABLE [dbo].[SopimusOsapuoli]
DROP CONSTRAINT Fk_SopimusOsapuoliRooli_SopimusOsaPuoli_Rooli
GO

DROP TABLE [dbo].[SopimusOsapuoliRooli]
GO
```

Kuva 22 SQL skripti

Jotta saadaan ID: kentälle haluttu identity ominaisuus, joudutaan taulu rakentamaan uudestaan niin, että taulun SopimusOsapuoliRooli_ID kolumnille annetaan IDENTITY arvo INT IDENTITY arvolla (Kuva 23).

```
CREATE TABLE [dbo].[SopimusOsapuoliRooli] (
    SopimusOsapuoliRooli_ID INT IDENTITY(1, 1) NOT NULL,
    Selite nvarchar(50) NOT NULL );
    GO

ALTER TABLE [dbo].[SopimusOsapuoliRooli]
ADD CONSTRAINT PK_SopimusOsapuoliRooli PRIMARY KEY (SopimusOsapuoliRooli_ID)
GO

SET IDENTITY_INSERT [dbo].[SopimusOsapuoliRooli] ON
GO
```

Kuva 23 SopimusOsapuoliRooli -taulun luonti

Luotiin taulu jossa on kaksi kolumnia, SopimusOsapuoliRooli_ID sekä Selite. Taulun ID kentästä tehtiin taulun PRIMARY KEY, jotta tauluun ei pystytä lisätä rooleja joilla on sama ID arvo. Nyt ID kolumnilla on IDENTITY ominaisuus, eli jos tauluun lisätään Selite, sille generoidaan automaattisesti ID arvo. Järjestelmän omat rooli ID:t halutaan alkavan nollasta, ja viimeinen järjestelmän rooli olisi kolme. Taululle ei voi itse syöttää ID arvoja, joten joudumme laittamaan taulun IDENTITY_INSERT:in päälle, jotta voimme syöttää manuaalisesti järjestelmän omat roolit.

```
INSERT INTO [dbo].[SopimusOsapuoliRooli] ([SopimusOsapuoliRooli_ID], [Selite])
    VALUES (0, 'Velallinen'), (1, 'Myöntäjä'), (2, 'Saatavan omistaja'), (3, 'Muu velallinen')
    GO
```

Kuva 24 Taulun arvojen syöttäminen

Tauluun lisätään manuaalisesti halutut roolit ja roolien ID:t INSERT INTO komennolla (Kuva 24). Seuraavaksi tulisi vielä tehdä Selite kentässä indeksoitu kolumni, jotta järjestelmään ei pystytä käyttäjän toimesta lisäämään rooleja, jotka ovat samoja kun järjestelmän omat. Indexointi onnistuu CREATE UNIQUE INDEX komennolla (Kuva 25).

```
CREATE UNIQUE INDEX UQ_SopimusOsapuoliRooli_Selite ON dbo.SopimusOsapuoliRooli (Selite)
GO
```

Kuva 25 UNIQUE INDEX

Jäljellä on enään taulun viiteavainten lisääminen niihin tauluihin, joista ne aikaisemmin pudotettiin (Kuva 22), sekä uniikkien ID arvojen generointi. Viiteavaimen tarkoitus on estää toiminnot jotka hajottaisivat suhteita tietokanta taulujen välillä. Viiteavain on kenttä tai kolumni taulussa, johon pääavaimella toisesta taulusta voidaan viitata (W3schools c). Viiteavaimet lisätään kahden taulun välille ADD CONSTRAINT komennolla (Kuva 26).

```
ALTER TABLE [dbo].[SopimusOsapuoli]
ADD CONSTRAINT FK_SopimusOsapuoliRooli_SopimusOsaPuoli_Rooli
FOREIGN KEY (Rooli) REFERENCES [dbo].[SopimusOsapuoliRooli](SopimusOsapuoliRooli_ID)
GO

ALTER TABLE [dbo].[TuoteOsapuoli]
ADD CONSTRAINT FK_TuoteOsapuoli_SopimusOsapuoliRooli
FOREIGN KEY (SopimusOsapuoliRooliId) REFERENCES [dbo].[SopimusOsapuoliRooli](SopimusOsapuoliRooli_ID)
GO
```

Kuva 26 Viiteavaimen lisäys

Viimeisenä täytyy tehdä toiminnallisuus joka generoi uudet ID arvot. Microsoft tarjoaa SQL serverille toiminnon nimeltä DBCC CHECKIDENT, joka mahdollistaa taulun index kolumnien arvojen manipuoloinnin. Sillä voidaan asettaa taulun identity kolumni alkamaan tietystä arvosta, joka sopii toteutukseen erittäin hyvin. Syötimme jo järjestelmän omat roolit sekä niiden ID arvot, joten voimme määritellä DBCC CHECKIDENT:in avulla, että kaikkille seuraaville uusille riveille aloitetaan ID arvot 1000 eteenpäin.

```
DBCC CHECKIDENT ('[dbo].[SopimusOsapuoliRooli]', RESEED, 999);
GO
```

Kuva 27 DBCC CHECKIDENT

CHECKIDENT ottaa erilaisia argumentteja mitä identity arvoille tehdään, mutta päätin käyttää RESEED argumenttia. RESEED:in avulla määritellään, että tämän taulun identity kolumnia muutetaan. Identity kolumneilla on vakiona +1 kasvu, joten antamalla RESEED:ille arvo 999, seuraava rivi joka tauluun syötetään, saa ID arvon 1000 (Kuva 27).

7 Yhteenveto

Toteutuksen tuloksena Profit Lainat ja Vakuudet -antolainajärjestelmä on monipuolisempi kuin ennen. Toteutus saatiin toteutettua vaatimusten mukaan, käyttöliittymä on yhdenmukainen muun sovelluksen kanssa (Kuva 27), sekä se helpottaa ja nopeuttaa roolien hallintaa. Uusi ala-otsikko sijaitsee halutussa paikassa, eli Työkalut -osion alapuolella omana ala-otsikkona. Uusia rooleja pystyy lisäämään, sekä lisättyjä muokkaamaan. Järjestelmän omat roolit ovat disabloituja, jotta niitä ei pysty muokkaamaan. Tietokanta tehtiin uudestaan niin, ettei järjestelmään pysty syöttämään siellä jo olevia rooleja, ja järjestelmään lisätyt roolit alkavat numerosta 1000, jotta ne eivät sekoitu jo olemassa oleviin järjestelmän omiin rooleihin.

Haluttu lisäominaisuus sovellukselle hyväksyttiin katselmoijien ja testaajien toimesta, ja julkaistiin seuraavassa tuotepäivityksessä osaksi olemassa olevaa sovellusta. Profit Lainat ja Vakuudet -antolaina järjestelmästä tuli näin monipuolisempi ja arvokkaampi loppukäyttäjille. Toteutus sai kiitettävää palautetta asiakkaalta ja tuotteen omistajalta.

Työn tekeminen opetti paljon ketterästä ohjelmistokehityksestä, sekä erittäin monipuolisesti ohjelmoinnista. Ohjelmointia tehtiin monipuolisesti ohjelmistokehityksen kaikilta osa-alueilta, ja tuntemus ohjelmoinnista sekä projektin tavoista ja arkkitehtuurista kasvoi toteutuksen myötä.

Lähteet

Sähköiset

Vainu. Profit Software Oy. https://vainu.io/company/profit-software-oy-taloustiedot-ja-liike-vaihto/267864/yritystiedot

Jyväskylän Yliopiston Informaatioteknologian tiedekunta. React. https://ap-pro.mit.jyu.fi/tiea2120/luennot/react/

Wikipedia a. https://fi.wikipedia.org/wiki/Devops

Wikipedia b. https://en.wikipedia.org/wiki/Front-end_web_development

Wikipedia c. https://fi.wikipedia.org/wiki/TypeScript

Wikipedia d. https://fi.wikipedia.org/wiki/Ohjelmointirajapinta

Wikipedia e. https://en.wikipedia.org/wiki/C_Sharp_(programming_language)

Dagmar. https://www.dagmar.fi/verkkopalvelukehitys/mita-markkinoijan-tulee-ymmartaa-web-ohjelmoinnista/

TypeScript https://www.typescriptlang.org/

Fellowmind. https://www.fellowmindcompany.com/fi-fi/ajankohtaista/kettera-ohjelmistokehitys-ja-sen-menetelmat/

Taskmill. https://taskmill.fi/koulutus/tuoteomistaja-product-owner-koulutus-sertifiointiin-tahtaava/

Scrum. https://www.scrum.org/resources/what-is-a-sprint-retrospective

Itwiki https://www.itewiki.fi/opas/laadunvarmistus-ja-ohjelmistotestaus/

Helsingin Yliopisto. https://fullstackopen.com/osa5/end_to_end_testaus

Profit a. https://profitsoftware.com/tuotteet/profit-loans-collaterals/?lang=fi

Cypress. https://docs.cypress.io/guides/getting-started/writing-your-first-test#Step-3-Click-an-element

Valjas. https://valjas.fi/opi/blogi/mita-integraatio-rajapinta-ja-api-tarkoittavat/

Microsoft a. https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet

Microsoft b. https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/

Mui. https://mui.com/material-ui/getting-started/learn/

Section. https://www.section.io/engineering-education/how-to-implement-material-ui-in-react/

Redux. https://react-redux.js.org/introduction/why-use-react-redux

React a. https://reactjs.org/docs/introducing-jsx.html

Tiko Jamk. https://tiko.jamk.fi/~tuito/websk20/frontendsk/index.html

W3schools a. https://www.w3schools.com/whatis/whatis_react.asp

W3schools b. https://www.w3schools.com/tags/ref_httpmethods.asp

W3schools c. https://www.w3schools.com/sql/sql_foreignkey.asp

W3schools d. https://www.w3schools.com/react/react_jsx.asp