VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Saku Lauttamus

QUERY BUILDER -SOVELLUKSEN KÄYTTÖ-LIITTYMÄTOTEUTUS

Case Fliq Oy

Liiketalous
2021

VAASAN AMMATTIKORKEAKOULU
Tietojenkäsittely

TIIVISTELMÄ

Tekijä	Saku Lauttamus
Opinnäytetyön nimi	Query Builder -sovelluksen käyttöliittymätoteutus Case Fliq Oy
Vuosi	2021
Kieli	suomi
Sivumäärä	36
Ohjaaja	Raija Tuomaala

Tämän opinnäytetyön tarkoituksena oli ohjelmoida käyttöliittymätoteutus toimeksiantajan asettamien tavoitteiden ja kriteereiden mukaisesti, uudelle ominaisuudelle nimeltään Query Builder. Koska tarkoituksena oli toteuttaa vain käyttöliittymä, jouduttiin varsinaisen back end-toteutuksen ja tietokannan sijaan käyttämään vaihtoehtoista ratkaisua hyödyntäen simuloitua tietokantadataa, jotta käyttöliittymällä esiintyvät visuaaliset elementit voitiin rakentaa funktionaalisuudeltaan loogisesti.

Ensimmäisen osion tarkoituksena on avata lukijalle projektin lähtöasetelma, esitellä toimeksiantaja, projektissa käytetyt työkalut sekä avata lukijalle käsitteet, joita varsinaisen projektitoteutuksen läpikäynnin yhteydessä esiintyy.

Toinen osio käsittelee projektin kulun suunnitteluvaiheesta siihen pisteeseen, kun todettiin toteutuksen olevan valmis. Tässä osiossa käydään projektin kulku vaiheittain tarkasti askel askeleelta, sekä havainnollistetaan lukijalle kuvin ja selityksin, millainen prosessi Query Builder -ohjelman rakentaminen oli PowerPoint -piirroksesta internetselaimessa käytettävään ohjelmaan saakka.

Kolmas osio opinnäytetyöstä koostuu yhteenvedosta, jossa tuodaan esille projektin lomassa koetut onnistumiset ja kehittämistarpeet sekä työn ohessa esiintyneet haasteet.

Avainsanat Angular, käyttöliittymäohjelmointi, typescript, html

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Business Information Technology

ABSTRACT

Author	Saku Lauttamus
Title	Query Builder -application's user-interface implementation Case Fliq Oy
Year	2021
Language	Finnish
Pages	36
Name of Supervisor	Raija Tuomaala

The objective of this thesis was to develop a user interface for a new feature called Query Builder as an element of Fliqs software portal. Since the goal was to create a fully functional user interface without developing the back end itself. There was a need for alternative implementation that allowed the database itself to be simulated to get the visible elements to function as logically as intended.

The purpose of the first section is to open to the reader the starting point of the project, the tools that were used within the project as well as introduce the company Fliq as a client and to define the concepts that are appearing within the thesis.

The second section of the thesis introduces the project itself. It takes the reader through the process step by step and illustrates the project flow with pictures and explanations. The process of developing Query Builder from PowerPoint drawing into functional web-application.

The third section consists of a summary of the process, which highlights the successes, the challenges experienced during the project and highlights the needs of development that can be still carried out after this project was finished.

Keywords Angular, user interface programming, typescript, html

SISÄLLYS

KUVALUETTELO

1 JOHDANTO

Tämän opinnäytteen tavoitteena ja tarkoituksena on tuoda esiin käyttöliittymän toteuttamisen eri vaiheita, sekä esittää lukijalle vaihtoehtoinen tapa rakentaa käyttöliittymäkokonaisuus ilman konkreettista tietokantaratkaisua. Lisäksi lukijalle on tarkoitus avata varsinaisen toteutusprosessin kulku käyttäen Angular Frameworkia käyttöliittymäkoodin luomisessa.

Angular lukeutuu kolmen käytetyimmän web-ohjelmointitekniikan joukkoon perinteisen JavaScriptin sekä Facebookin kehittämän React Frameworkin rinnalle. Nykyaikana web-ohjelmointi on yleistynyt niin merkittävästi, että ohjelmoijan työkalupakkiin on miltei välttämättömyys kuulua osaamista vähintään yhdestä edellä mainituista sovelluskehitystekniikoista tai sovelluskehyksistä.

Projektin aloitusvaiheessa omaan jo hieman entuudestaan opittuja perusteita Angularin suhteen, mutta toteutuksen edistyessä uskon tietotaitoni ennestään kasvavan. En ole aiemmin toteuttanut sovellusta ilman konkreettista tietokantaa, saati sovellusta, jonka funktionaalisuuteen vaikuttavat tekijät eivät olleet jokaisessa tapauksessa ennalta määritettyjä. Toiminnallisuus saattoi muuttua ns. dynaamisesti sovellusta käytettäessä. Näin ollen sovelluksen tuli reagoida eri tavalla tilanteisiin, joiden lopputulema määräytyi suhteessa käytön yhteydessä annettuihin parametreihin.

Aihe opinnäytetyölle valikoituu osana intoa oppia uutta web-kehityksen saralla, sekä ajatuksesta syventää osaamistani nykyaikaisempien web-kehitysmenetelmien osalta tosielämän työtilanteissa. Henkilökohtaisena tavoitteena opinnäytetyöprojektissa minulla on oppia rakentamaan edistyneempi sovellus, jonka käyttötarkoitus olisi käytännöllinen oikeassa yritys- tai työmaailmassa.

2 FLIQ PROJEKTIN TOIMEKSIANTAJANA

Fliq Oy on vaasalainen sovelluskehitykseen erikoistunut yritys, jonka pääsääntöisenä tarkoituksena on kehittää ja tarjota tuotettaan Fliq. Fliq on tuotteena modulaarinen sovellusalusta, joka on ensisijaisesti tarkoitettu käytettäväksi teollisuuden kesken. Tuotetta tarjotaan siitä kiinnostuneille asiakkaille, sekä tuotetta räätälöidään tapauskohtaisesti kunkin asiakkaan tarpeiden mukaisesti. Fliq tarjoaa käyttäjälleen työkalun, johon on sisällytetty mm.

- erilaisia hallinta- ja suunnittelutyökaluja
- matkaviestintävälineiden, erillisten koneiden, työnkulun ja IOT-laitteiden etävalvonnan mahdollistavia työkaluja
- Lean -valmistyökalut

Tämän kaiken yhtenäisenä tavoitteena on mahdollistaa asiakkaan liiketoiminnan tehostaminen entisestään. (Fliq Oy 2020.)

Ensipuraisuni web-kehittämiseen sekä ohjelmoinnin harjoittamiseen ammattikäyttöisessä työympäristössä alkoi Fliq Oy:n kanssa yhteistyönä koulutukseen osana kuuluvan harjoittelujakson myötä. Harjoittelujakson loputtua aloin työskentelemään Fliq:lle opiskelujen ohessa, ja sen innoittamana päätin kysyä, voisinko toteuttaa opinnäytetyöprojektini yrityksen toimeksi antamana. Seuraava askel olikin päättää, millainen projekti tulisi olla luonteeltaan, jotta se olisi kuitenkin kyllin laaja, mutta silti toteutettavissa kesän pituisen ajanjakson puitteissa. Muutamien viikkojen sisäisen pohdinnan ja keskustelujen jälkeen, saatiin aihe lyötyä lukkoon, ja näin aiheeksi valikoitui tämä Query Builder -työkalu. Työkalun tarkoituksena on visualisoida järjestelmään kirjattua dataa siten, että esimerkiksi erillisiä töitä, tai työvaiheita voidaan verrata keskenään. Näistä voidaan koota dataa koskien vaikkapa sitä, miten monta työtuntia tietty työvaihe vie tietyn työkokonaisuuden kohdalla, tai mikä vaihe on yleisesti aikaa vievin tai muuta vastaavaa. Koska käytettävä data haetaan järjestelmän kesken, tämän ominaisuuden tavoitteena oli myös kyetä tarkastelemaan tapahtumia reaaliajassa, jolloin jokapäiväinen työnseuranta helpottuu entisestään työnjohdossa oleville henkilöille.

3 TOTEUTUKSESSA KÄYTETYT TYÖKALUT JA KESKEISIMMÄT KÄSITTEET

Toteutettaessa ohjelmointikokonaisuuksia, käytössä saattaa olla useampiakin erilaisia teknologioita ja työkaluja. Tämän projektin kohdalla suurinta roolia näyttelivät Visual Studio Code, Angular Framework versio 8, sekä versionhallintatyökalu Git. Tässä kappaleessa on tarkoituksena avata lukijalle hieman, millaisista työkaluista on kyse, ja miten ne liittyivät projektiin pitkin toteutuksen eri vaiheita. Tarkoituksena on myös avata toteutusprosessin läpikäynnissä esiintyviä keskeisimpiä käsitteitä.

3.1 Keskeisimmät työkalut, joita toteutuksessa käytettiin

3.1.1 Microsoft Visual Studio Code

Visual Studio Code (jatkossa VS Code) on Microsoftin tarjoama ilmainen koodieditori, joka tarjoaa toimivan ja monipuolisen ympäristön WEB-ohjelmointiin. VS Code on käytettävyydeltään hyvin kevyt, eikä näin ollen tarvitse tietokoneelta suurta määrää resursseja toimiakseen optimaalisesti. VS Code on myös käännetty universaaliksi siinä mielessä, että sitä voidaan käyttää tarvittaessa Windows-, macOS- sekä Linux-käyttöjärjestelmillä. Tämä tekee siitä erittäin hyvän työkalun käytettäväksi mm. ohjelmointitiimien kesken. VS Codesta tekee erityisen tehokkaan työkalun web-ohjelmointiin liittyen se, että siihen on sisällytetty myös sisäänrakennettu tuki JavaScriptille, node.js:lle sekä typescriptille. Näiden sisäänrakennettujen ominaisuuksien lisäksi, VS Code omaa valtavan kokoelman erilaisia laajennuksia (extensions), joita käyttäjä voi valikoida tarpeen mukaan käytettäväksi ja täten optimoida editoria tarvitsemaansa käyttöön rajatakseen päinvastaisesti myös pois haluamiaan laajennuksia ja ominaisuuksia. Visuaalinen sekä funktionaalinen mukautettavuus on myös toteutettu erittäin laajasti editorissa, käyttäjän on mahdollista muokata fontteja, asetteluja, fonttien värejä, erilaisia hotkey-näppäinyhdistelmiä oman mielen mukaisesti lähes loputtomiin. (Visual Studio Code 2020.) Tämä editori on myöskin yksi suosituimmista web-kehittäjien käyttämistä editoreista, joka

käytännössä takaa editorin toimivuuden vielä pitkälle tulevaisuuteen (DesignLibb 2020).

3.1.2 Angular-framework

Angular on HTML ja Typescript koodikieliä hyödyntävä sovelluskehys, jonka avulla kyetään luomaan nykyaikaisia web-sovelluksia, ja verkkosivuja. Angularia käytettäessä ohjelmoija kykenee itse vaikuttamaan tahtomaansa lopputulokseen käyttämällä juuri niitä Typescript -kirjastoja, joista kulloinenkin käyttötarkoitus hyötyy parhaiten, tarjoamalla näin ohjelmoijalle laajan alustan luoda erilaisiin käyttötarpeisiin räätälöityjä sovelluskokonaisuuksia. Angular -sovelluskehyksen avulla verkkosivujen ja -sovellusten luonti perustuu usean erillisen Angular komponentin luontiin, jotka luokitellaan `ngModuleiksi`. Komponentit voivat esimerkiksi määrittää näkymät, jotka ovat rakenteeltaan joukko näyttöelementtejä, joita Angular voi valita käsitelläkseen tavoin, jotka ohjelmoija on määrittänyt koodissaan muun logiikan ja toiminnallisuuden yhteydessä. Komponentit hyödyntävät myös niin kutsuttuja palvelu -osia, jotka voivat tarjota toiminnallisuuksia, jotka eivät suoraan ole sidoksissa yksittäisiin näkymiin. Näin ollen palveluita voidaan importoida dependencyinä eli eräänlaisina riippuvuuksina käytettäväksi laajalti sovelluksessa, tehden koodista uudelleenkäytettävää, tehokasta ja modulaarista. (Angular 2021.)

3.1.3 Git

Git on tällä hetkellä maailman eniten käytössä oleva versionhallintajärjestelmä, sekä aktiivisesti ylläpidetty avoimen lähdekoodin projekti, jonka kehittäjänä toimii Linux-käyttöjärjestelmästäänkin tuttu Linus Torvalds. (BitBucket 2021.) Git on käytännössä näitä kolmea asiaa: ohjausjärjestelmä, versionhallintajärjestelmä ja hajautettu versionhallintajärjestelmä. Ohjausjärjestelmänä Git on käytännössä sisällönjäljittäjä/-kirjanpitäjä. Gitiä voidaan käyttää paikkana, johon tallentaa ja varastoida sisältöä, yleisesti tämä sisältö on jotenkin sidoksissa ohjelmointiin ja on mitä todennäköisimmin koodia, sillä mm. koodin varastoinnin lisäksi Git tarjoaa muita loistavia ominaisuuksia, kuten versionhallintajärjestelmän. Koodi, joka on säilötty Gitiin päivittyy sitä mukaa, kun siihen lisätään uusia rivejä näin ollen koodia voidaan muuntaa niin yksin kuin tiimin toimesta. Jotta muutoksista voidaan pitää

kirjaa, on oltava jonkinlainen tapa tehdä niin, tämän tavan tarjoaa Git versionhallintajärjestelmänä. Gitin yksi vaikuttavimpia ominaisuuksia ovat ns. haarat eli branchit, joiden kautta voidaan käyttää pohjana yhtä toimivaa ohjelmoitua kokonaisuutta (master), ja sen pohjalta lisätä siihen kunkin ohjelmoijan toimesta ominaisuuksia(feature-branch) turvallisesti ilman, että satunnaisten lisäysten myötä jokin olemassa oleva toiminnallisuus vaurioituisi tai muuttuisi toimintakyvyttömäksi. Tälläisiä brancheja kutsutaan ominaisuus- eli feature-brancheiksi, ja vastaavasti näiden lähtöpistettä master-branchiksi. Tämä ominaisuus helpottaa mm. testaamista merkittävissä määrin sillä, kun feature-branch yhdistetään master-branchiin, voidaan rakentaa testiympäristöön päivitetty versio. Mikäli se ei toimi, voidaan paikantaa Gitin toimintalokista minkäniminen haara on yhdistetty ns. isäntähaaraan ennen kuin sovelluksen toiminta vaurioitui ja perua tämä yhdistäminen sovelluksen toiminnan säilyttämiseksi. Hajautetuksi versionhallintajärjestelmäksi Gitiä kutsutaan siksi, koska koodi on hajautettuna jokaisen sitä käsittelevän ohjelmoijan tietokoneille. Git omaa ns. etätietovaraston, joka on tallennettuna pavelimelle, ja paikallinen tietovarasto, joka taas on tallennettuna kunkin ohjelmoijan tietokoneelle. Tämä mahdollistaa sen, että alkuperäinen koodi ei ole tallennettuna vain keskuspalvelimelle, vaan alkuperäisen koodin täydellinen kopio on myös kaikkien kyseistä koodia muokkaavien ohjelmoijien tietokoneilla. (Aditya Sridhar FreeCodeCamp 2018.)

3.2 Käsitteet

3.2.1 Back End

Termillä back end viitataan tietokonemaailmassa kaikkiin niihin verkko-ohjelmiston tai verkkosivuston osiin, jotka eivät ole konkreettisesti näkyvillä käyttäjälle. Yleisesti nimitystä back end käytetään puhuttaessa ohjelmiston tai sivuston niin sanotusta tiedonsiirtokerroksesta, jonka sisäisesti tiedonsiirto tapahtuu tiedonlähteen ja visuaalisen käyttöliittymän välillä. (Per Christensson TechTerms 2020.)

3.2.2 Mock data

Mock data tarkoittaa suomennettuna mallidataa. Tällainen data lisätään keinotekoisesti ohjelmiston tai järjestelmän sisään, jotta sillä voidaan tapauskohtaisesti simuloida malliskenaarioita ilman, että tarvittaisiin luoda varsinaista tietokantarelaatiota tai tiedonsiirtokerrosta. Mallidatan suurimpina hyötyinä mainittakoon sen rakennettavuuden yksinkertaisuus verrattuna lopulliseen tietokantaratkaisuun, sekä nopea muunneltavuus tilannekohtaisesti. Haittapuolena kuitenkin on se, että ilman riittävää tuntemusta ohjelmistoa kohtaan, on myös helppo luoda tapauksia, joissa tietoa käsitellään tosielämässä vastaan tulevien tapausten vastaisesti. (Rosie Hamilton 2016.)

3.2.3 Datasource

Datasource, eli sanasanaan suomennettuna tietolähde, on tietojenkäsittelyn yhteydessä paikka, johon sovelluksen käyttämä data on varastoitu, ja jonka kautta kyseistä dataa käsitellään. Ensisijaisesti ohjelmistoissa tietolähteenä käytetään konkreettista tietokantaa, mutta esimerkiksi opinnäytetyön tapauksena, tietolähteenä käytetään ohjelmiston yhteydessä kovakoodattua dataa. (Technopedia 2020.)

3.2.4 Front End

Termillä front end viitataan tietokonemaailmassa vuorostaan kaikkiin niihin verkko-ohjelmistojen tai verkkosivustojen osiin, jotka ovat käyttäjälle konkreettisesti näkyvissä. Front end koodi sisältää mahdolliset käyttöliittymän asettelut, sekä kaiken sen koodin, joita silmin nähtävät toiminnallisuudet ohjelmassa vaatii. Siinä missä back end käsitteenä kattaa tiedonsiirtokerroksen ohjelmistorakenteessa, front end kattaa kaikki ne osa-alueet, jotka ovat visuaalisesti näkyvissä ohjelmiston tai sivuston käyttäjälle. (Per Christensson TechTerms 2020.)

3.2.5 Typescript

Typescript on Javascript -ohjelmointikieleen perustuva avoimen lähdekoodin ohjelmointikieli. Javascriptiin pohjautuen Typescript tarjoaa kaikki ne ominaisuudet mitä Javascript, sisältäen lisäominaisuutena oman tyyppijärjestelmänsä.

Tyyppijärjestelmän ansiosta koodi pitää jatkuvasti huolen siitä, onko luodut tyypit määritetty koodissasi ja näin ollen vähentää merkittävästi määrittämättömien tyyppien seurauksena ilmentyvien bugien määrää. (TypeScript 2021.)

3.2.6 HTML

HTML on web-sivujen luomiseen käytettävä standardi ohjelmointikieli. HTML on lyhenne englannin kielen sanoista Hyper Text Markup Language, karkeasti suomennettuna; hypertekstin merkintäkieli. HTML:n tarkoituksena on kuvata verkkosivun rakenteellisuus, ja se toimii ikään kuin suuntaviivana selaimelle kertoen, miten verkkosivun sisältö tulee käyttäjälleen näyttää. (W3Schools. 2021.)

3.2.7 Treeview

Treeview- nimitys tulee englannin kielestä ja suora suomen kielen käännös sille on puunäkymä, jonka tarkoituksena on yksinkertaistaa hierarkkisesti näytettävän tietokokonaisuuksien visuaalista esiintymistä käyttöliittymällä. (WinDev. 2020.)

3.2.8 Metodi

Metodi on yleisesti luokkaan liitetty proseduuri/menettely tai toiminto, jota käytetään pääsääntöisesti osana olio-ohjelmointia (object-oriented programming). Metodin pääsääntöinen tarkoitus on määritellä luokan käyttäytyminen tietyn instanssin yhteydessä. (Technopedia. 2012.)

3.2.9 Funktio

Funktion on koodikielen yksikkö, jonka määritelmä perustuu sen rooliin osana koodin rakennetta. Funktio on siis erilaisista koodin palasista rakentuva kokonaisuus, jonka toiminta perustuu erilaisiin koodaajan itsensä antamiin syötteisiin (input), jotka pyritään prosessoimaan funktion avulla konkreettisiksi sovelluksen toiminnassa käytettäviksi arvoiksi tai toiminnallisiksi ominaisuuksiksi. (Technopedia. 2017.)

3.2.10 Parametri

Parametrillä tai argumentilla tarkoitetaan ohjelmoinnin yhteydessä jotain sellaista arvoa, joka välitetään funktiolle työstettäväksi halutun tuloksen saamiseksi. (TechTerms 2014.) Kuten esimerkiksi jos funktiokutsussa tahdottaisiin funktion tulostavan jotain parametreillä X ja Y, on käyttäjän annettava välitettäväksi sisällöksi jokin arvo parametrien X ja Y mukaisesti. Vaikkapa ”X = Hello;” ja ”Y = World;”, jotta funktio tulostaisi lausahduksen Hello World. (Per Christensson TechTerms 2020.)

4 KÄYTTÖLIITTYMÄN TOTEUTUS

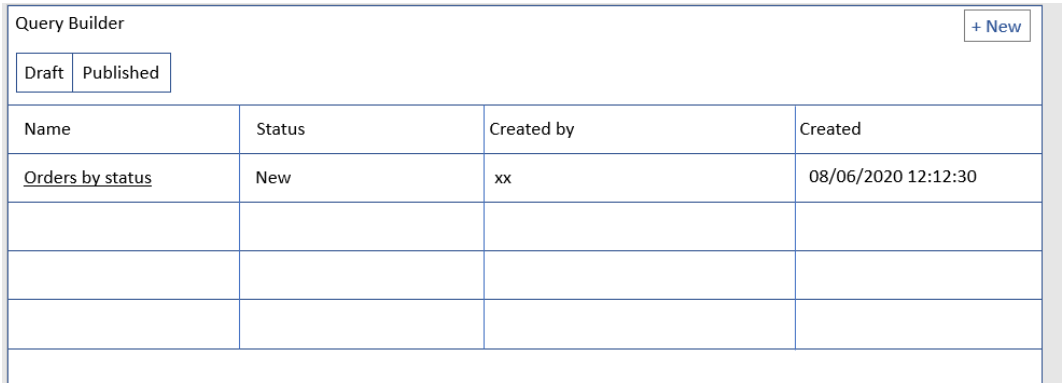
Tässä opinnäytetyössä toteutettiin käyttöliittymäratkaisu asiakastilauksena tulleelle uudelle ominaisuudelle nimeltään Query Builder. Seuraavissa kappaleissa käydään läpi varsinaisen projektin kulku, sekä havainnollisestaan projektissa luodun käyttöliittymän toiminnallisuutta.

4.1 Ideointi

Projekti aloitettiin ajatuksen pohjalta toimeksiantajani kanssa jo varhain huhtikuussa 2020. Ajatuksena oli rakentaa ja integroida uusi ominaisuus jo olemassa olevaan sovellusportaaliin hyödyntämällä järjestelmästä jo entuudestaan löytyviä komponentteja ja moduuleja.

4.2 Suunnittelu

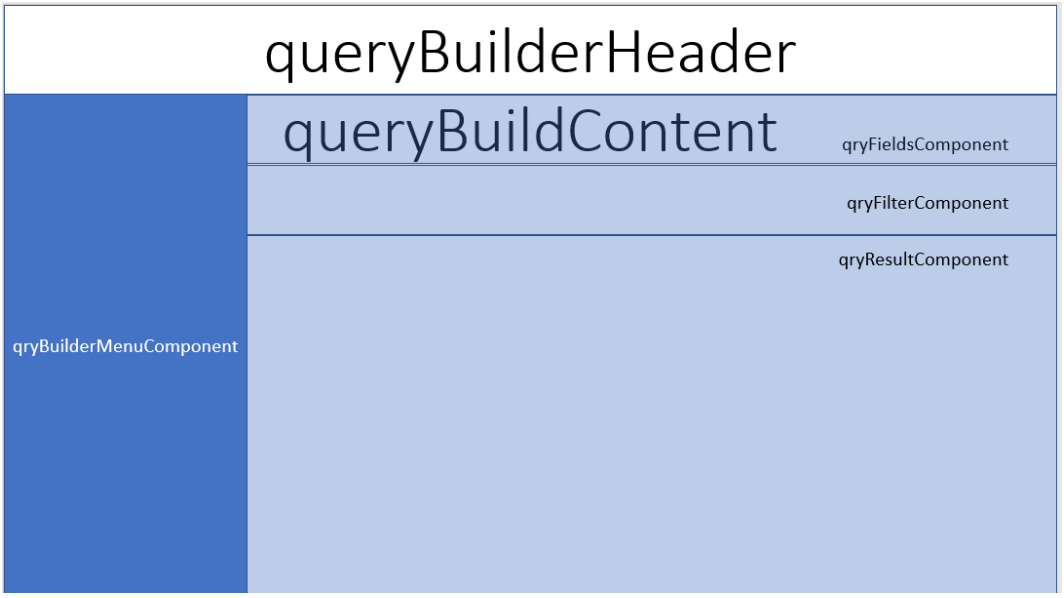
Toimeksiannon vahvistuttua, ja projektin aloitusajankohdan selkeydyttyä, alettiin suunnittelemaan tulevan uuden komponentin rakennetta. Suunniteltu ajankohta projektin toteutukselle oli kesä-elokuun 2020 välisenä aikana. Komponenttia koskien pidettiin noin tunnin mittainen palaveri, jossa käytiin läpi toimeksiantajan toive komponentin toiminnallisuudesta, käyttötarkoituksesta sekä ajatukset aloitusvaiheesta. Palaverissa rakennettiin myös reaaliaikaisesti ensimmäiset suunnittelua koskevat piirrokset käyttämällä työkalua Microsoft PowerPoint.



Kuva 1. Kuvankaappaus sovelluksen ensimmäisestä rakennusvaiheesta.

Sovelluskomponentin päänäkymän rakenne hahmottuu kuvasta seuraavasti. Vasemmalla ylhäällä näkyy sovelluksen otsikko eli header. Oikeaan yläkulmaan on sijoitettu nappi, jonka tarkoitus on ohjata sovellus luonti- eli create-näkymään. Otsikon alapuolelle on asetettu suodatusta kuvaavat painikkeet ”Draft” ja ”Published”, joiden toiminnallisuus on sidottu alapuolella näkyvään table- eli taulu- näkymään. Taulun sisältö koostuu tallennetun kyselyn perusarvoihin, jotka tullaan asettamaan kyselylle sovelluksen create-näkymässä.

Palaverin aikana käytiin läpi myös tuleva muokkaus- eli edit-näkymä, josta luotiin mallinnus päänäkymästä tehdyn suunnitelman mukaisesti.



Kuva 2. Kuvankaappaus sovelluksen edit-näkymän rakenteen mallista

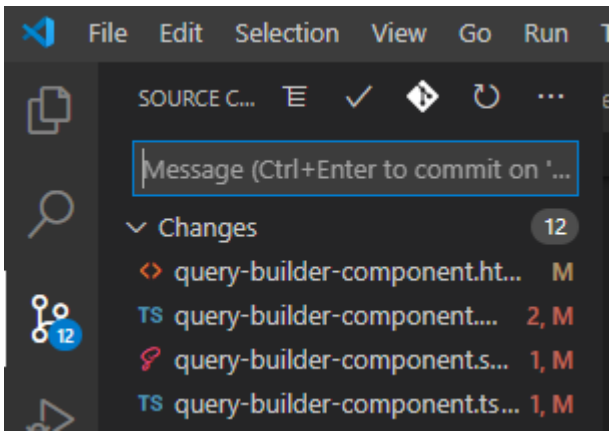
Kuva 3. Tarkennettu kuvaus sisältäen havainnollistavia tietoja toiminnallisuudesta.

Kuvista hahmottuu näkymässä käytettävät ydinkomponentit, kuten ”qryBuilder-MenuComponent”, joka sisältää puunäkymän eli treeview:n. Treeview:n sisään asetetaan datakokonaisuuksia kuten kuvassa osoitettu ”Service Order”, joita käytetään myöhemmin osana sovelluksen tavoiteltua funktionaalisuutta. Datakokonaisuuksia kutsuttakoon tässä tapauksessa nimellä treenode. Edellä mainittuja treenodeja hyödyntäen, osaksi funktionaalisuutta liitetään mukaan seuraava kenttä qryFieldsComponent. Tämän kentän tarkoituksena on suodattaa tietokannasta haettavaa dataa qryResultComponent:n sisältämään taulunäkymään. Suodattaminen tapahtuu käyttäjän raahatessa treenode-elementti puunäkymästä qryFieldsComponent-elementin päälle ja tapahtuman eli eventin seurauksena suodatin saa parametrin Service Order. Parametrin asetuttua data hakeutuu tietokannasta kyseisen parametrin mukaisesti ja tulostuu taulunäkymään.

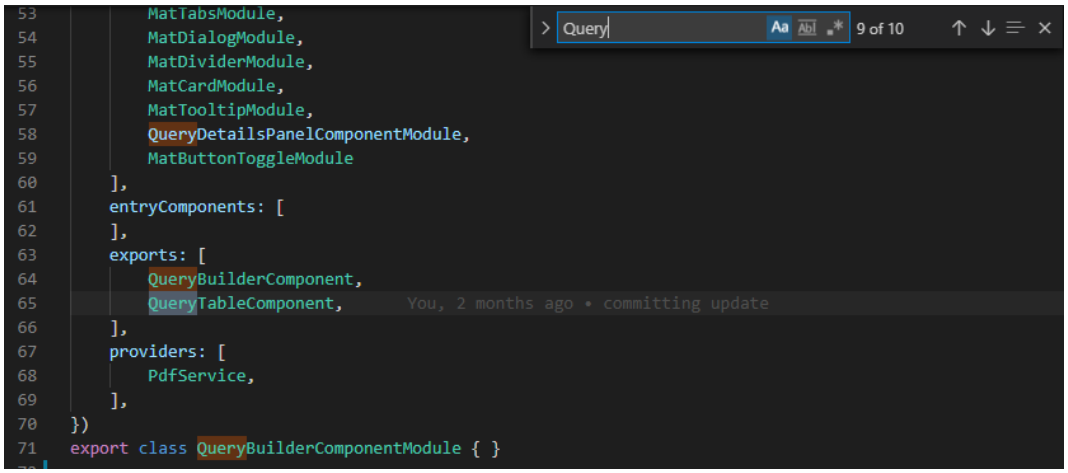
Seuraavaksi käytiin nopea suullinen keskustelu siitä, mistä lähdetään rakentamaan sovellusta. Mitä valmiita sovelluksia voidaan käyttää pohjana tai mallina niiden sisältämien, samankaltaisten ominaisuuksien mukaan. Tämän jälkeen palaveri päätettiin ja varsinainen sovelluksen rakentaminen aloitettiin.

4.3 Toteutus

Toteutuksen ensimmäisessä vaiheessa valittiin valmis sovellus, jonka toiminnallisuus vastaisi parhaiten tulevan Query Builder -sovelluksen toivottuja ominaisuuksia. Mallisovellukseksi valikoitui valmis `Tasks-module` -moduuli, jonka kansiot kopioitiin ja nimettiin uudelleen vastaamaan uuden moduulin nimeä. Seuraavana työvaiheena kopioidun `Tasks-module` tiedostot käytiin läpi, ja nimettiin uudelleen jokainen funktio ja metodi, joka sisälsi viitteitä task-moduuliin. Uudelleen nimeämisessä käytettiin hyödyksi VS Coden ominaisuutta: Source control. Tämän ominaisuuden johdosta navigoiminen tiedostojen välillä on huomattavasti helpompaa, sillä branchiin tehtäessä muutoksia, kyseisen välilehden alla näkyvät vain tiedostot, joihin on tehty muutoksia poiketen alkuperäisestä versiosta. Näin ollen kyettiin avaamaan tiedosto ja korvaamaan jokainen tiedoston sisältämä Task/Tasks uudella Query/Queries arvolla käyttäen apuna etsi- työkalua.



Kuva 4. Kuvankaappaus VS Coden tarjoamasta Source control työkalusta



Kuva 5. Kuvankaappaus Etsi/Find- työkalusta ja visualisointi sen tarjoamasta helpotuksesta korvattavia arvoja etsittäessä.

Uudelleennimeämisprosessin jälkeen uudelle moduulille annettiin luokka `QueryBuilderComponentModule`, jonka määrittäminen on edellytys moduulin liitettävyydelle osaksi sovelluskokonaisuutta. Luokan määrityksen jälkeen, luokka exportoitiin eli tuotiin käytettäväksi universaalisti sovelluksen sisällä lisäämällä komento: `export class QueryBuilderComponentModule {}` tiedostoon `query-builder-component.ts`. Seuraavaksi luokka importoitiin `base-component.module.ts` -tiedostoon, jonka kautta luokkaa ja sen sisältämiä määrityksiä kutsutaan siirryttäessä käyttämään Query Builder sovellusta.

```
const appRoutes: Routes = [
    {
        path: 'main',
        component: BaseComponent,
        canActivate: [AuthGuard],
        children: [
        { path: 'query-builder', pathMatch: 'full', loadChildren: () => import('../query-builder/query-builder-component.module').then(m => m.QueryBuilderComponentModule) },
        { path: 'query-form', pathMatch: 'full', loadChildren: () => import('../query-builder/query-form/query-form-component.module').then(m => m.QueryFormComponentModule) },
    ]
}
```

Kuva 6. Rajattu kuvankaappaus moduulin importoinnista sovelluskokonaisuuteen.

Kun sovellus saatiin käytettäväksi ja näkyviin osana käyttöliittymää, siirryttiin muovailemaan sovelluksen päänäkymää. Siihen tuli kuvan 1. mukaisesti luoda taulu, jossa näkyisi käyttäjien luomat kyselyt (Queries). Tässä työvaiheessa käytettiin hyödyksi jo olemassa olevaa task-table -taulurakennetta liittämällä se osaksi `query-builder`-sovelluskomponenttia. Kyseinen komponentti huomioitiin uudelleennimeämisprosessissa ja nimettiin uudelleen sisältöineen `query-table`:ksi.

Seuraavaksi `query-table` importoitiin varsinaiseen `query-builder-component`-moduuliin, sekä varmistettiin arvojen olevan `query-table` mukaiset `query-builder-component`:n html-tiedostossa. Tämän jälkeen rakennettiin sovellus uudelleen syöttämällä komento `npm run build` VS Coden terminaaliin ja rakennusprosessin päätyttyä kyettiin havaitsemaan, että taulu ilmestyi osaksi käyttöliittymää. Tässä vaiheessa tauluun ei ollut vielä asetettu minkäänlaista dataa liikuteltavaksi suuntaan tai toiseen.

4.4 Aloitusnäkymän luonti

Käyttöliittymä luotiin siihen pisteeseen, että siihen tuli rakentaa jonkinlainen dataratkaisu. Tässä vaiheessa päädyttiin ratkaisuun, jolla kyettiin simuloimaan varsinaisesta tietokannasta kulkeutuvaa dataa niin kutsutun mock datan avulla. Sovelluksella on käytössään servicekirjasto, jota hyödynnetään back endin ja front endin välisen tiedonkulun välietappina. Tiedonsiirtokutsut, jotka on määritetty back endillä voidaan kääntää muotoon, jota Angular kykenee ymmärtämään osana funktioita. Tässä tapauksessa, kun back end -osuutta ei rakennettu varsinaisen tietokannan päälle, luotiin mock dataa eli tapauskohtaista kovakoodattua dataa suoraan luokan sisään servicekirjastossa, sekä luotiin olennaiset funktiot tiedonsiirtoa varten. Servicen luonti tehtiin samalla periaatteella, kuin moduulikin. Ensin luotiin luokka `QueryMockService`, joka myöhemmin exportoitiin ja importoitiin moduulikohtaiseen käyttöön, aivan kuten komponentti sovellus- tai moduulikohtaiseen käyttöön. Tämän jälkeen luotiin itse dataa sisältävä array, jonka arvot määritettiin tapahtumakohtaisesti ui-koodin mukaisiksi.

```
public mockData: any[] = [
  {
    query_id: '1',
    query_name: 'Some test data',
    query_description: 'testings',
    created_by: 'User1',
    date: '22.06.2020',
  },
  {
    query_id: '2',
    query_name: 'Some good name',
    query_description: 'test n.o:2',
    created_by: 'User2',
    date: '22.06.2020',
  },
];
```

Kuva 7. Kuvankaappaus mock data-arrayn rakenteesta.

```
setTableColumns(): void {
    this.queryTableColumns = [
        { def: 'select', type: TableCellType.MULTISELECT, width: '5%' },
        { def: 'query_name', name: 'Query name', filter: { type: 'input' }, sortable: true, width: '20%' , type: TableCellType.LINK },
        { def: 'query_description', name: 'Description', langTag: 68, filter: { type: 'input' }, sortable: true , width: '20%' },
        {
            def: 'created_by', name: 'Created by:', langTag: 399, sortable: true, width: '20%' ,
            filter: { type: 'select', data: this.userFilterSelectList, filteredData: this.userFilterSelectList }
        },
        { def: 'date', name: 'Date created:', langTag: 334, filter: { type: 'date' }, sortable: true, width: '20%'  },
    ];
}
```

Kuva 8. Kuvankaappaus funktiosta, joka määrittää arvot käyttöliittymällä esiintyvän taulun sarakkeille.

Kun mock data-array taulua varten saatiin rakennettua, asetettiin taulun sarakkeet luovan funktion arvot datasourcemme mukaisiksi, jotta saatiin oikea data-arvo sitoutumaan oikean sarakkeen kanssa. Kuten kuvista käy ilmi, mock dataan asetettujen arvojen tulee olla samat, kuin arvot joita funktio `setTableColumns` asetettiin käyttämään. Muussa tapauksessa funktio ei kyennyt hakemaan dataa ja kyseinen sarake tai koko sarakeryhmä tuli jäämään tyhjäksi. Nyt, kun käyttöliittymälle tulostuvaan tauluun asetettiin arvot sarakkeille, oli aika luoda funktio, joka hakee taulussa näkyvien rivien sisältämän datan. Tämä funktio sijoittuu serviceen data-arrayn alapuolelle, ja sen tarkoituksena on hakea `get`-metodin omaisesti yksittäinen `slice()` eli aaltosulkeiden{} sisältämä yksittäinen dataobjekti array-taulusta.

```
/* Function to get mocked data */
getMockData(): Observable<any> {
  return of(this.mockData.slice());
}
```

Kuva 9. Kuvankaappaus datanhakufunktiosta.

Nyt oltiin tilanteessa, jossa datasource on määritetty, sekä funktio hakemaan tätä kyseistä dataa ja siirryttiin importoimaan serviceä käytettäväksi moduulin koodissa. Tässä vaiheessa käytettiin pakettimanageria npm, jotta saatiin luotua riippuvuus- eli dependency-paketti kyseisestä servicestä sovelluksen build-vaihetta varten. Antamalla VS Coden konsoliin komento `npm run package` saatiin luotua paketti järjestelmäpolkuun, joka viittasi kloonattuun git-repositoryyn jossa Fliq-sovelluksen servicekirjasto sijaitsi työasemallamme. Seuraavaksi paketti asennettiin editorissa, jossa ui-koodi oli auki syöttämällä komento: `npm i` *`järjestelmäpolku johon paketti luotiin`*. Tämä sama prosessi suoritettiin aina, kun service-tiedostoon tehtiin muutoksia, jotta uusin versio servicestä saatiin käytettäväksi. Kun uusi dependency-paketti saatiin asennettua, siirryttiin taulua koskevaan `query-table-component.ts` -tiedostoon ja muutettiin siellä sijaitsevan `ngOnInit()` -funktion sisältöä niin, että datanhaku tapahtui juuri luodusta `queryMockServicestä` hyödyntäen sieltä `getMockData`-funktiota.

```
ngOnInit(): void {
    this.queryMockService.getMockData().subscribe(data => console.log(data));
    console.log(this.columns);
}
```

Kuva 10. Kuvankaappaus ngOnInit-funktiosta. Funktioon lisättiin myös dataliikennettä kuvaava lokikomento: `console.log()`. Tämän komennon avulla kyettiin näkemään konkreettisesti selaimen konsoli-ikkunasta mitä dataa kyseinen funktio hakee.

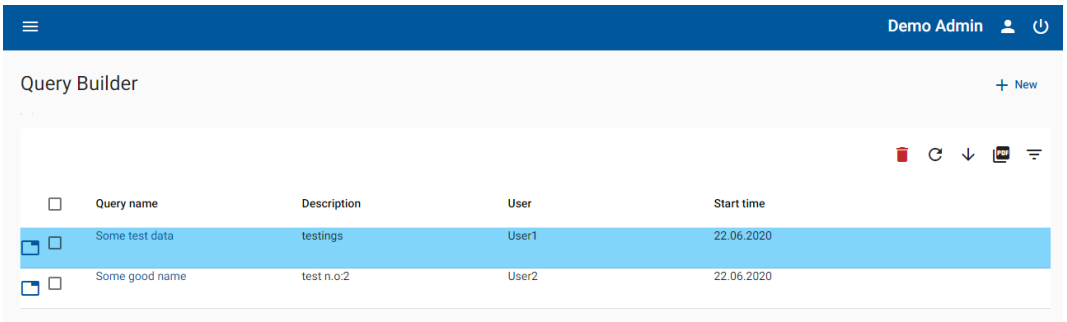
Lopuksi, jotta tauluun saatiin näkymään juuri luomamme data tuli asettaa `[data-source]` arvo oikein taulun rakennetta koskevassa html-tiedostossa, sekä saattaa aiemmin mainitsemamme `setTableColumns` -funktio käytäntöön liittämällä se osaksi `query-builder-component.html` -tiedostoa.

```
<app-table [dataSource]="queryMockDataSource" [pageSize]="pageSize" [pageSizeOptions]="pageSizeOptions" [columns]="columns"
      [records]="records" [delete]="true" [refresh]="true" [exportPDF]="true" [exportExcel]="true"
      [sortCol]="'query_id'" [rows]="queryMockDataSource" [detailCol]="true" (onMarkCheck)="selectRow($event)"
      (onDelete)="deleteQuery($event)" (onRefresh)="refreshTable($event)" (onExportPDF)="exportQueryPDF($event)"
      (onCellClick)="cellClick($event)" (onExportExcel)="exportTable($event)"
      (onOpenDetailPanel)="openDetailPanel($event)"></app-table>

<query-table *ngIf="filtersReady" [columns]="queryTableColumns" [node]="node" [nodeType]="nodeType"
    [periods]="periodFilterSelectList" [nodeId]="nodeId" [filters]="filters"
    (onCellClick)="cellClick($event)" (onMarkCheck)="saveToArray($event)"
    (onDeleteRows)="deleteQueries($event)" (onOpenDetailPanel)="openDetailPanel($event)">
</query-table>
```

Kuva 11. Kuvankaappaus `[datasource]` arvon määrityksestä taulun rakennekoodissa, sekä kuvankaappaus query-builder-componentin kohdasta, jossa sarakkei-den arvo `[columns]` määritetään `setTableColumns` -funktion sisään määritetyn arvon mukaiseksi.

Nyt datasource oli asetettu, samoin taulun sarakkeet ja oli aika ajaa komento: `npm run build` ja rakentaa uusi versio käyttöliittymä-koodista, sekä testata sitä selaimella. Onnistuneen uudelleenrakennusprosessin jälkeen käyttöliittymä näytti hyvin pitkälti tältä.



Kuva 12. Kuvankaappaus valmistuneesta aloitusnäkymästä sisältäen luodun mock-datan.

4.5 Lomakkeiden luominen

Nyt kun käyttöliittymään oli luotu aloitusnäkymä, oli aika liittää siihen toiminnallisuutta sen verran, että saataisiin luotua siihen uutta tietoa. Tämä työvaihe osoittautui huomattavasti yksinkertaisemmaksi, sillä käytännössä mallina käytetyssä moduulissa oli valmiina luontilomake (form) asetteluineen. Seuraavana työvaiheena tuli siis karsia valmiista formista turhat kentät pois, joita ei katsottu tälle moduulille olennaisiksi. Moduulille olennaiset kentät määräytyivät suoraan mock data-arrayn sisällön mukaisesti, sillä näiltä kentiltä siirtyvä tieto tultaisiin liittämään

myöhemmässä vaiheessa osaksi simuloitua tietokantaa. Muutosten jälkeen, query-formin html -koodi näytti seuraavan kuvan mukaiselta.

```
<form *ngIf="queryForm" [formGroup]="queryForm" (ngSubmit)="submitForm()" padding-horizontal style="padding: 20px" style="height: 100%;">
    <div class="query-form-content container" style="height: 100%; width: 100%;">
        <div fxLayout="column">
            <div fxLayout="row" fxLayoutGap="15px">
                <span class="query-form-header">{{header}}</span>
            </div>
        </div>
    <div class="query-details" *ngIf="formType === 'new'">
        <div fxLayout="column">
            <!-- Query name -->
            <mat-form-field class="query-form-name">
                <input matInput placeholder="{{166 | langTag}}" formControlName="query_name">
            </mat-form-field>
        </div>
        <div fxLayout="column">
            <!-- Description  -->
            <mat-form-field class="query-form-descr">
                <textarea matInput placeholder="{{68 | langTag}}"
                    formControlName="query_description"></textarea>
            </mat-form-field>
        </div>
        <div fxLayout="column">
        <!-- User  -->
            <mat-form-field class="query_user">
                <input matInput placeholder="Created by:"
                    formControlName="created_by">
            </mat-form-field>
        </div>
        <div fxLayout="column">
            <!-- Date -->      You, a few seconds ago • Uncommitted changes
            <mat-form-field class="query-form-date">
                <input matInput placeholder="Date created:" formControlName="date">
            </mat-form-field>
        </div>
    <div fxLayout="row" fxLayoutAlign="end" style="padding-right: 16px;">
        <!-- Cancel -->
        <button mat-button type="button" (click)="cancel()">{{275 | langTag}}</button>
        <!-- Save -->
        <button mat-raised-button color="accent" type="submit">{{279 | langTag}}</button>
    </div>
</div>
```

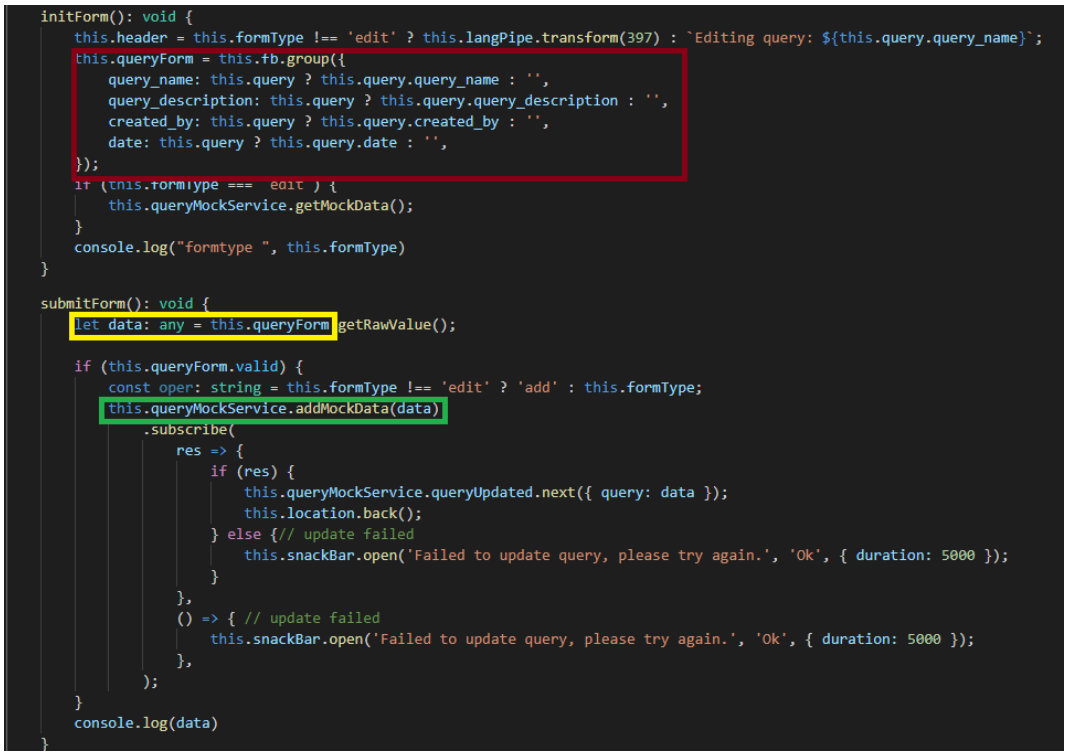
Kuva 13. Kuvankaappaus New-tyyppisen query-formin html-tiedostosta.

Seuraavassa työvaiheessa, kun formin HTML-tiedosto muokattiin toivotun lopputuleman mukaiseksi, kirjoitettiin serviceen uusi funktio lisäämään data-slice aiemmin luomaamme mock data-arrayhin.

```
addMockData(data: any): Observable<any> {
  this.mockData.push(data);
  return of(true);
}
```

Kuva 14. Kuvankaappaus rakennetusta lisäysfunktiosta `query.mock.service`-tiedostossa.

Seuraavaksi uusi luotu funktio valjastettiin käyttöliittymä-koodin käyttöön seuraavasti.



```
initForm(): void {
    this.header = this.formType !== 'edit' ? this.langPipe.transform(397) : `Editing query: ${this.query.query_name}`;
    this.queryForm = this.fb.group({
        query_name: this.query ? this.query.query_name : '',
        query_description: this.query ? this.query.query_description : '',
        created_by: this.query ? this.query.created_by : '',
        date: this.query ? this.query.date : '',
    });
    if (this.formType === 'edit') {
        this.queryMockService.getMockData();
    }
    console.log("formtype ", this.formType)
}

submitForm(): void {
    let data: any = this.queryForm.getRawValue();

    if (this.queryForm.valid) {
        const oper: string = this.formType !== 'edit' ? 'add' : this.formType;
        this.queryMockService.addMockData(data)
            .subscribe(
                res => {
                    if (res) {
                        this.queryMockService.queryUpdated.next({ query: data });
                        this.location.back();
                    } else {// update failed
                        this.snackBar.open('Failed to update query, please try again.', 'Ok', { duration: 5000 });
                    }
                },
                () => { // update failed
                    this.snackBar.open('Failed to update query, please try again.', 'Ok', { duration: 5000 });
                },
            );
    }
    console.log(data)
}
```

Kuva 15. Kuvankaappaus serviceen luodun funktion käytöstä käyttöliittymä-koodin joukossa.

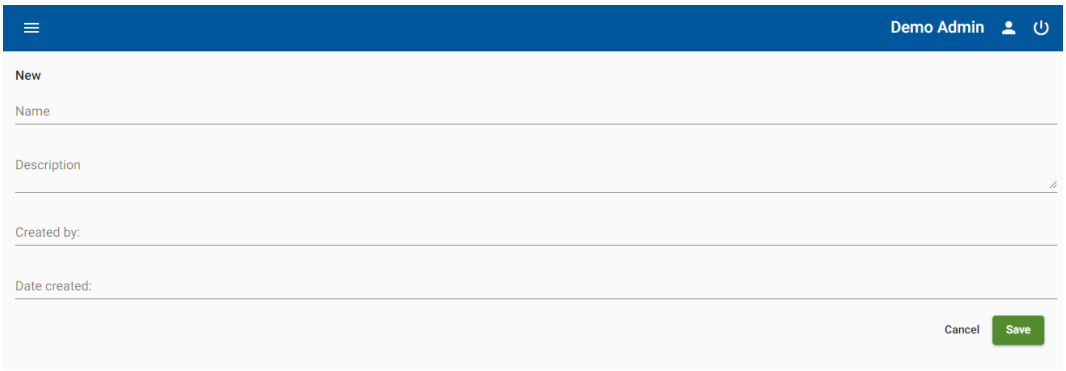
Punaiseen laatikkoon sisälletyllä alueella muodostettiin `initForm` -funktion käynnistyksen yhteydessä muuttujalle: `queryForm` arvo `this.fb.group`, eli ryhmän verran data-arvoja. Nämä kyseiset arvot tulevat myöhemmin kattamaan sen, mitä käyttäjä tulee asettamaan saman nimisiin html-kenttiin formia käyttäessään formin itsensä tyypistä riippumatta.

Keltaisen laatikon korostamassa kohdassa hyödynnetään `initForm` -funktion aikana kutsuttua muuttujaa `queryForm` sisältöineen ja näin ollen voidaan asettaa arvo(t) parametrille: data, kun käyttäjä laukaisee funktion `submitForm`.

Vihreä laatikko havainnollistaa `addMockData` -funktion käyttöä tilanteessa, jolloin formin tyyppi ei ole `edit` ja toiminto `add` käynnistetään. Tarkoitus on myös havainnollistaa servicekirjastoon luodun funktion yhtenäinen toiminta muun käyttöliittymä-koodin joukossa.

Tässä vaiheessa uuden tiedon luomiseen käytettävä formi saatiin luotua ja se oli käyttövalmis. Seuraavaksi oli tietenkin testattava sitä käytännössä, ja rakennettava

käyttöliittymä uudelleen syöttämällä terminaaliin komento `npm run build` ja asentaa myös muokattu service-tiedostopaketti uudelleen kuten tehtiin aloitusnäkymän `getMockData` -funktion, sekä mock data -tietokannan kanssa. Testi voitiin todeta onnistuneeksi, ja seuraavaksi oli muokattava editointiform toimivaan muotoon, joka prosessina itsenään oli aluksi aivan vastaavanlainen, kuin uuden querytiedon luominen. Ainoana käytännön erona oli varmistaa formin tyypin vaihtuminen oikean tapahtumasarjan seurauksena ja tässä tapauksessa se oli tuplaklikkaamalla taulussa näkyvää riviä. Koska mallina toimineen moduulin toiminta oli asetettu edellä mainituksi, ei tämän tarkastaminen vaatinut erityisempiä toimenpiteitä.



Kuva 16. Kuvankaappaus valmiista luonti-ikkunasta.

4.6 Muokkauslomakkeen viimeistely

Nyt moduulin perustoiminnot olivat siinä pisteessä, että kyettiin toteuttamaan siihen lisäominaisuuksia toimeksiantajan toiveiden mukaisesti. Seuraavaksi kävimme läpi, mitä saatiin aikaan ja miten moduuli toimi tällä hetkellä, jonka jälkeen siirryttiin lisäohjeistukseen moduulin muokkausnäkymän toteutuksen viimeistelemiseksi. Tämä työvaihe oli projektin hankalin, sillä toteutuksen onnistumiseksi täytyi käsitellä dynaamisesti tuotettavaa tietotaulua, sekä luoda sen kanssa yhteen toimiva suodatuskenttä, jotta käyttäjä saisi taululle tulostumaan vain tapauskohtaisesti haluamaansa dataa. Sovelluksessa itsessään oli valmiiksi toteutettu puunäkymä, josta oli mahdollista jalostaa tulevaa muokkauslomaketta varten parametrilista, josta drag and drop -toimintoa käyttäen käyttäjä kykenisi valikoimaan haluamansa suodatusparametrin myöhemmin taululle tulostuvalle datalle. Ensimmäinen vaihe oli paikantaa valmiiksi toiminnassa oleva `tree-module`, jotta se saatiin kopioitua,

sekä muokattua tarpeen vaatimaan muotoon. Tämä edellytti lähinnä nimeämismuutoksia käyttöliittymä-koodin puolelta, mutta servicen puolelle sen sijaan jouduttiin toteuttamaan yksi valtava array -taulukko sisältäen kaikki puuhun kiinnitettävät `treenode`:t ja niiden kautta näkyvissä olevan datan. Array -taulun lisäksi luotiin myös get-funktio, jotta saatiin haettua data sitä hyödyntämällä käyttöliittymä-komponentille.

```
public MockTreeData: any[] =[
{
  name: 'Jobs',
  children: [
    {
      name: 'Job Name',
      li_attr: {
        col: 'job_name',  // column definition
        col_name: 'Job name'  // column name
      }
    },
    {
      name:'Description',
      li_attr: {
        col: 'job_desc',
        col_name: 'Description'
      }
    },
```

Kuva 17. Kuvankaappaus kahdesta puunäkymässä esiintyvästä vaihtoehdosta. Array konaisuudessaan käsitti tapauksessa 16 kappaletta saman rakenteen omaavaa vaihtoehtoa.

Treeview komponentin säädöksien jälkeen importoitiin se jälleen osaksi pääkomponenttia `query-builder`:ia. Tämän importoinnin jälkeen, muutettiin hieman edit formin asettelua, jotta saatiin sivusta toivotun näköinen. Muokkausnäkymä asettelultaan tehtiin aluksi vastaamaan uuden luontia ja näin ollen tehtiin mm. muutos kenttien asetteluun siten, etteivät ne enää olleet kaikki allekkain, vaan kentät: `Name, Created By` ja `Date Created` sijoitettiin vierekkäin sivun yläosaan. Niiden alle sijoitettiin `Description` -kenttä yhtä leveänä kuin yllä olevat kolme. Näin saatiin kenttien alle lisää tilaa myöhemmin rakennettavaa taulua sekä suodatuskenttää varten. Seuraavaksi kenttien alapuolelle sijoitettiin `<div>` -html-

elementti, jonka sisällä hyödynnettiin ominaisuutta `flexbox`. Tämä ominaisuus helpotti erinäisten html-elementtien sijoittelua div-elementin sisällä, ja mahdollisti halutun ulkoasun toteuttamisen viimeisille kolmelle komponentille formin sisällä. Treeview-komponentti sijoitettiin div-elementin `query-content` sisään vasempaan reunaan, jonka jälkeen luotiin komponentti hyödyntäen Angular elementtiä: `mat-chip-list`. Tämä komponentti toimii myöhemmin lisättävän taulun suodatuskenttänä. Html-tiedostossa komponentille annettiin lähinnä visuaalisesti vaikuttavia arvoja pois lukien sen tyyppi `cdkDropList`. Tämän komponentin toiminta määritettiin tarkemmin muun projektin luonteisesti `.ts` eli typescript-tiedostossa seuraavalla tavalla.

```
allowDrop(evt: DragEvent): void {
    evt.preventDefault();
}

drop(event: CdkDragDrop<any>) {
    if (event.previousContainer === event.container) {
        moveItemInArray(event.container.data, event.previousIndex, event.currentIndex);
    } else {
        transferArrayItem(event.previousContainer.data,
            event.container.data,
            event.previousIndex,
            event.currentIndex);
    }
    this.draggedNodes.push(event.container.data[0]);
}
//Remove event for mat-chip in dropzone.
remove(node: Node): void {
    const index = this.nodes.indexOf(node);
    if (index >= 0) {
        this.nodes.splice(index, 1);
    }
}
```

Kuva 18. Kuvankaappaus suodatus komponenttiin liittyvistä funktiosta.

Funktion `allowDrop` tarkoituksena on käännöksensä mukaisesti sallia raahatun elementin pudottaminen kursorin osoittamalle alueelle. `Drop` -funktion tarkoituksena on mahdollistaa funktion `moveItemInArray/transferItemInArray` sisältämien parametrien mukaisen datan liikuttaminen edelliseltä container-elementiltä toiselle, joko siirtämällä osiota (item) array taulun sisällä, tai viemällä se arraystä toiseen. Tämän jälkeen `event.container.data` -array:hin sisällytetään myöhemmin osana datataulun toiminnallisuutta käytettävä data-arvo `draggedNodes`. `Remove` -funktio liittyy mat-chip-elementin sisäiseen toimintaan mahdollistaen kyseisen elementin poistamisen näkyvistä filter-komponentin sisästä. Tämä

funktio valjastettiin käyttöön yksittäisen mat-chipin sisällä html-tiedostossa, jossa mat-chip:lle annettiin arvo `[removable] = 'removable'`. Näin komponentti: mat-chip-list saatiin toimimaan yhdessä drag and drop -ominaisuuden kanssa, sekä suodattimelle annettujen arvojen poistaminen mahdollistettiin ilman, että sivu täytyisi ladata uudelleen arvojen nollaamiseksi.

Seuraavaksi oli aika viimeistellä komponentti, sekä lisätä siihen sen viimeinen osa eli datataulu. Tämän datataulun tarkoituksena on näyttää dataa perustuen valintoihin, joita käyttäjä tekee aiemmin mainittujen puunäkymän, sekä suodatuspalkin avulla. Tässä kohdassa hyödynnettiin valmiiksi rakennettua taulukomponenttia `query-table-component`, mutta edellisestä poiketen taulun sarakkeita ei asetettu manuaalisesti `setTableColumns()` funktion yhteydessä, sillä dynaamisesti rakennetulle datalle jouduttiin rakentamaan omat funktionsa. Uuden sarakkeiden asetus -funktion lisäksi, täytyi muodostaa uusi mock data-array, jotta kyettiin hakemaan kulloiseenkin tapaukseen relevanttia dataa taululle.

```
export class mockTableData {

  mockTableRecords: number = 0;

  public mockJobsData: any[] = [
    {
      job_name:'job1',
      job_desc:'job1desc',
      job_number:'1',
      worker_organisation:'organisation1',
      asset:'asset1',
      date_created:'21.8.2020',
      created_by:'tester',
    },
  ]
```

Kuva 19. Kuvankaappaus uudesta mock data-arraystä, sekä yhdestä sen sisältämästä dataobjektista.

Taulun itsensä ulkoasu otettiin suoraan aiemmin luodusta, mutta uutta käyttötarkoitusta silmällä pitäen siitä kuitenkin karsittiin muutamia ominaisuuksia mm. editointilomakkeen aukeaminen riviä klikkaamalla ja taulurivin poistaminen. `QueryMockService` -tiedostoon lisättiin neljä uutta samankaltaista data-array:tä

kuin aloitusnäkymän taulun yhteydessä, jotta saataisiin jokaiselle puunäkymän vaihtoehdolle oma yksilöllinen data-array myöhempää toiminnallisuutta varten. Poiketen aloitusnäkymän tauluun liittyvistä service muutoksista, tällä kertaa rakennettiin vain get-funktio jokaista erillistä arrayta kohden, sen sijaan, että rakennettaisiin myös add- ja remove-funktiot. Kyseisen taulun sisältämää dataa ei tässä näkymässä ollut määrä muuttaa, lisätä tai poistaa, jonka vuoksi kyseiset funktiot olisivat olleet keskeiselle toiminnalle turhia. Seuraavaksi lisättiin näkymään nappi, jonka avulla kyettäisiin hakemaan suodatinpalkkiin asetettujen data-arvojen muodostaman parametrin mukaisesti oikea get-metodi näistä neljästä uudesta vaihtoehdosta. Jotta kyseisistä get-metodeista saatiin tilanteeseen käytännön hyötyä, luotiin myös funktio, jonka avulla saatiin rakennettua taululle tulostuvan tiedon mukaiset sarakkeet siten, että sarakkeilla lukevat otsikot määräytyisivät array:hin asetettujen otsikkojen mukaisesti.

```
setQueryFormTableColumns(nodeData: any[]): void {
    this.queryFormTableColumns = [];
    for (const node of nodeData) {
        this.queryFormTableColumns.push({ def: node.li_attr.col, name: node.li_attr.col_name });
    }
}
```

Kuva 20. Kuvankaappaus metodista, jonka avulla saatiin muunnettua treenodejen sisältämä data käännettyä taulun sarakkeiden muodostamista varten.

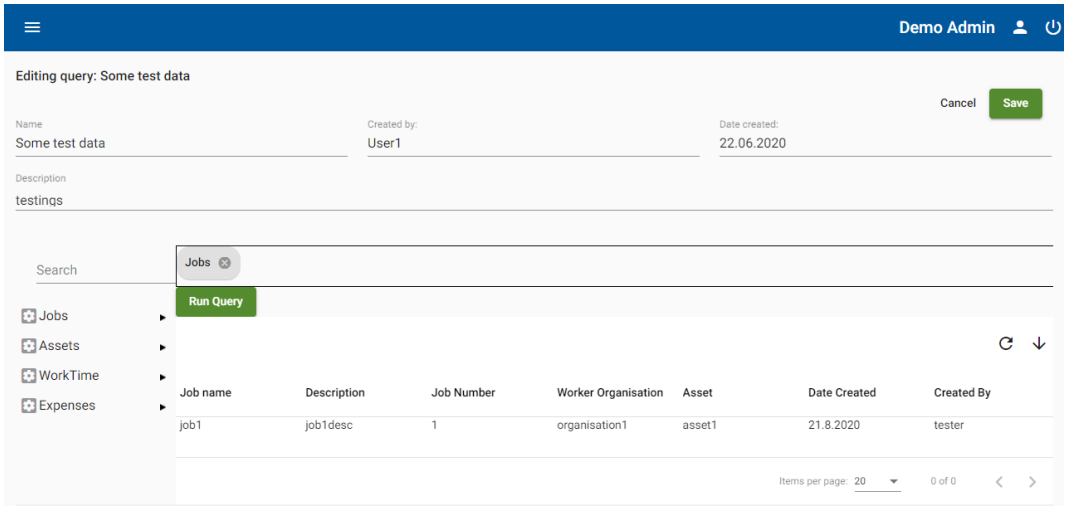
`SetTableColumns`-funktion avulla treenodejen sisältämä data, tässä tapauksessa Kuvan 14. mukainen data. Funktiossa luodaan yksittäinen node, nodeData-arraystä, jonka jälkeen jokaista arrayn sisältämää riviä kohden luodaan uusi `node.li_attr.` tauluun asetettavien sarakkeiden vaatimille arvoille def ja name. Näin saatiin `nodeData:`sta muunnettua `mockTreeData`, ja muunneettua kyseisen arraytaulun mukaiset otsikot myös otsikoiksi tulevan datataulun sarakkeille.

```
<button type="button" mat-raised-button color="accent" (click)="getQueryData()">Run Query</button>
```

```
getQueryData() {
    this.setQueryFormTableColumns(this.draggedNodes[0].children);
    this.loaded = true;
    if (this.queryFormTable) {
        this.queryFormTable.table.refreshTable();
    }
}
```

Kuva 21. Kuvankaappaus uudesta `<button>` elementistä (yläpuoli.). Sekä `getQueryData`-funktiosta(alapuoli).

`GetQueryData`-metodin tehtävänä on kääntää suodatuspalkkiin annetun array:n: `draggedNodes[0].children` sisältö parametreiksi metodin `setQuery-FormTableColumns` käyttöön, jotta taulun sarakkeet saadaan luotua sen mukaan, mitä käyttäjä on suodattimeen asettanut. Lopuksi oli aika testata lomaketta, ja todeta sen toimivuuden tila.



Kuva 22. Kuvankaappaus lopputulemasta, johon projekti päätettiin.

5 YHTEENVETO

Lähtökohtaisesti Angular -ympäristö oli minulle entuudestaan jokseenkin tuttu. Ennen projektia, en kuitenkaan ollut käsitellyt luotavaa dataa dynaamisesti eri käyttöliittymäkomponenttien välillä tai tehnyt suurempia käyttöliittymäkohtaisia ominaisuuksia tai kokonaisuuksia Angularilla. Mock datan luonti ja sen käyttö tuli toisena merkittävänä asiana, jonka kanssa aluksi olikin ongelmia. Haastetta tarjosi myös erinäiset komponenttien asetteluun liittyvät ongelmat mm. ennen flexboxiin liittyvän dokumentaation tarkempaa tutkimista. Jälkeenpäin mietittynä, monen asian olisi voinut hoitaa lähtökohtaisesti dokumentaatioita lukemalla sen sijaan, että yrittäisi saada ratkaisun muodostettua yrittämällä erilaisia ratkaisuja ns. ”via trial and error”. Mutta riittävän sitkeän tiedonhaun, sekä kollegoideni aktiivisen avustamisen ansiosta eteen tulleet haasteet oli mahdollista selättää vähintäänkin riittävän hyvin.

Projekti onnistui muuttuvan luonteensa, haastavan aikataulun, sekä vallitsevan koronavirustilanteen pakottaman etätyöskentelyn aiheuttamista haasteista huolimatta kuitenkin hyvin. Aikataulullisesti projektin kulusta teki hankalaa mm. tiedonkerääminen ja sen soveltaminen Internetistä sen sijaan, että olisi voinut kysyä suoraan kollegoilta, sillä projekti ajoittui hyvin pitkälti ajanjaksolle, jolloin oli kesälomat käynnissä. Lopputulokseksi saatiin kuitenkin kehityskelpoinen käyttöliittymä haluttuine ominaisuuksineen, mutta aivan lopulliseen muotoonsa sitä ei vielä saatu. Kehitettävää jäi mm. hienosäätöjä komponenttien asettelussa sivulla sekä niiden koon säätäminen responsiivisemmaksi suuremman näyttöresoluution ollessa valittuna. Suurimmat mainittavat kehityskohteet lienevät tietokantaan lisättävät taulut sekä back end toteutus varsinaista tietokantaa vasten.

Lähteet

Angular. Introduction to Angular concepts. 2021. Viitattu 3.2.2021. https://angular.io/guide/architecture#introduction-to-angular-concepts

Bitbucket. What is git. 2021. Viitattu 3.2.2021. https://www.atlassian.com/git/tutorials/what-is-git

Bitbucket. Why Git for your organization. 2021. Viitattu 3.2.2021. https://www.atlassian.com/git/tutorials/why-git

DesignLibb. List of Best Code Editor for Web developers. 2020. Viitattu 7.11.2020. https://designlibb.com/blog/list-of-best-code-editor-for-web-developers-2020/

FreeCodeCamp. An introduction to Git: what it is, and how to use it. 2018. Viitattu 3.1.2021. https://www.freecodecamp.org/news/what-is-git-and-how-to-use-it-c341b049ae61/¨

Fliq Oy. About Us. 2020. Viitattu 14.3.2021. https://www.fliq.fi/recruitment/

Scott logic. Data Mocking – A way to test the untestable. 2016. Viitattu 28.12.2020. https://blog.scottlogic.com/2016/02/08/data-mocking.html

Technopedia. Data Source. 2020. Viitattu 28.12.2020. https://www.techopedia.com/definition/30323/data-source

Technopedia. Function. 2017. Viitattu 3.2.2021. https://www.techopedia.com/definition/25615/function

Technopedia. Method. 2012. Viitattu 3.2.2021. https://www.techopedia.com/definition/3231/method

TechTerms. Backend. 2020. Viitattu 28.12.2020. https://techterms.com/definition/backend

TechTerms. Frontend. 2020. Viitattu 28.12.2020. https://techterms.com/definition/frontend

TechTerms. Parameter. 2014. Viitattu 3.2.2021. https://techterms.com/definition/parameter

TypeScript. TypeScript for JavaScript programmers. 2021. Viitattu 3.2.2021. https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html

Visual Studio Code. Getting Started. 2020. Viitattu 7.11.2020. https://code.visualstudio.com/docs

W3Schools. HTML Introduction. 2021. Viitattu 3.2.2021. https://www.w3schools.com/html/html_intro.asp

WinDev. The TreeView control. 2020. Viitattu 3.2.2021. https://doc.windev.com/en-US/?1013037&1013037