OULUN AMMATTIKORKEAKOULU

Tuukka Sarkkinen

MOBIILISOVELLUS KOKSAAMON OPERATIIVISEN TOIMINNAN TUEKSI

MOBIILISOVELLUS KOKSAAMON OPERATIIVISEN TOIMINNAN TUEKSI

Tuukka Sarkkinen
Opinnäytetyö
Kevät 2024
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma, ohjelmistokehitys

Tekijä: Tuukka Sarkkinen
Opinnäytetyön nimi: Mobiilisovellus koksaamon operatiivisen toiminnan tueksi
Työn ohjaajat: Hannu Ritola, Jouni Juntunen
Työn valmistumislukukausi ja -vuosi: Kevät 2024
Sivumäärä: 42

Opinnäytetyön toimeksiantajana toimi SSAB:n Raahen terästehtaan koksaamo. Koksaamolla oli tarve mobiilisovellukselle, josta saataisiin nopeasti tietoa prosessin tilasta. Opinnäytetyön tavoitteeksi tuli tehdä monialustainen mobiilisovellus tuotannolle, joka toimisi seurantatyökaluna reaaliajassa. Reaaliaikaisen seurannan avulla voitaisiin reagoida mahdollisiin häiriötilanteisiin nopeasti, jotta mahdollisilta tuotantokatkoksilta vältyttäisiin.

Työ tehtiin Microsoft Visual Studio Enterprisellä käyttäen .NET MAUIa sovelluksen rakentamiseen ja ASP.NET Core Web APIa ohjelmistorajapinnan luomiseen. Ohjelmointikielinä opinnäytetyön teossa käytettiin C#-ohjelmointia, jota käytettiin sovelluslogiikan toteuttamiseen sekä XAML-ohjelmointia käyttöliittymän rakentamiseen. Opinnäytetyön aineistot on kerätty aihealueeseen liittyviltä verkkosivuilta.

Työn tuloksena saatiin toimiva sovellus Android-laitteille ja siihen rakentui neljä eri sivua toiveiden mukaisesti. Etusivulle tuli näkyviin säätiedot ja muut sivut sisälsivät uunien purkuaikataulun, aamu- ja iltavuorojen purkujen ero optimiin taulukon sekä työntö- ja taakseteho maksimeiden pylväskaavion. Sovellus toimii tavoitteiden mukaisesti ja tiedot välittyvät sovellukseen nopeasti reaaliajassa. Tavoitteista jäätiin kuitenkin hieman, koska iOS-laitteille sovellusta ei toteutettu. Sovelluksen tekeminen iOS-laitteille olisi vaatinut erillisen lisenssin Applelle sekä MacOS-tietokoneen.

Asiasanat:
DOTNET MAUI, XAML, C#, Telerik, ASP.NET, Web API

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in information technology, option of software development

---

Author: Tuukka Sarkkinen
Title of thesis: Mobile application to support coking plant operational activities
Supervisors: Hannu Ritola, Jouni Juntunen
Term and year when the thesis was submitted: Spring 2024
Number of pages: 42

This thesis was commissioned by SSAB Raahe coking plant. The coke plant had a need for a mobile application to quickly obtain information about the process status. The aim of the thesis was to create a cross-platform mobile application for production, serving as a real-time monitoring tool. Real-time monitoring would enable quick responses to potential disruptions, in order to avoid possible production interruptions.

The work was carried out using Microsoft Visual Studio Enterprise to build the application with .NET MAUI and create the software interface with ASP.NET Core Web API. As programming languages, C# programming was used to implement the application logic and XAML programming to build the user interface. The materials for the thesis were gathered from relevant websites related to the subject area.

As a result of the work, a functional application for Android devices was obtained, consisting of four different pages as per to the wishes. The front page displayed weather information, while the other pages included a schedule for oven unloading, a table depicting the difference between morning and evening shifts unloading time to the optimum, and a bar chart of push and pull power maximum. The application operates according to the set goals, and data is transmitted to the application rapidly in real-time. However, there was a slight deviation from the goals since the application was not developed for iOS devices. Creating the application for iOS devices would have required a license from Apple as well as a MacOS computer.

Keywords:

DOTNET MAUI, XAML, C#, Telerik, ASP.NET, Web API

SISÄLLYS

1 JOHDANTO

Opinnäytetyön aiheeksi valittiin mobiilisovelluksen toteuttaminen Raahen SSAB:n koksaamon häiriötilanteiden ja operatiivisen toiminnan tueksi. Aiheen valinnan taustalla on saada koksaamon tuotannolle mobiilisovellus, josta saa nopeasti reaaliaikaista tietoa prosessin tilasta ja tärkeimmistä mittauksista, koska koksaamolla ei sellaista ennestään ole olemassa. Jos mittauksista havaitaan jatkuvia poikkeamia, voidaan niihin reagoida, ennen kuin niistä seuraisi mahdollisia tuotantokatkoksia. Tuotantokatkoset aiheuttavat ylimääräisiä kustannuksia yhtiölle.

1.1 SSAB:n Raahen tehdas

SSAB:n Raahen tehtaalla valmistetaan terästä ja sen päätuotteita ovat kuumavalssatut levyt ja kelatuotteet. Tehdas työllistää noin 2500 SSAB:n omaa työntekijää, joiden lisäksi satoja urakoitsijoiden ja yhteistyökumppaneiden edustajia. (SSAB 2023.) Raahen tehdas tunnettiin ennen nimellä Rautaruukki ja se perustettiin vuonna 1960. Vuonna 2014 SSAB hankki omistuksiinsa Rautaruukin osakevaihtojärjestelyin. (Wikipedia 2023.)

Raahen SSAB:lla sijaitsevalla koksaamolla valmistetaan koksia kivihiilestä, jota käytetään masuuneilla pelkistysaineena ja raaka-aineena brikettien valmistuksessa (Halonen 2013, 11). Koksaamon kahdella patterilla ja niiden 70 uunilla tuotetaan vuorokaudessa noin 2500 tonnia koksia (sama, 12). Koksaamon keskeisin osa on koksipatterit, joista ensimmäinen otettiin käyttöön vuonna 1987 ja toinen vuonna 1992 (Mannermaa 2012, 11). Ne ovat olleet toiminnassa koko ajan tähän päivään saakka. Koksipatterin uuneille tehdään panostus- ja purkutapahtumia noin 105 kappaletta vuorokaudessa. Koksausprosessissa syntyy merkittävänä sivutuotteena koksikaasua, jota hyödynnetään lämpöenergian lähteenä muissa prosesseissa Raahen tehtaalla (sama, 11). Koksin tuotanto on pitkä prosessi ja siihen liittyy monia eri vaiheita, joten mobiilisovellus antaa tuotannon työntekijöille työvälineen, jolla seurata esimerkiksi tärkeitä prosessiin vaikuttavia lämpötiloja ja tehoja.

1.2 Tavoitteet

Tämän opinnäytetyön ensimmäisenä tavoitteena on kehittää toimiva mobiilisovellus, joka vastaa koksaamon tarpeita. Tämä edellyttää sovelluksen suunnittelua, ohjelmointia ja testausta, jolla varmistetaan sen toimivuus ja luotettavuus. Tämä tavoite on ensisijainen, koska sovelluksen laatu ja toiminnot ovat keskeisiä tekijöitä.

Toisena tavoitteena on mahdollistaa prosessin reaaliaikainen seuranta kuvaajien ja taulukoiden avulla. Sovelluksen tulisi tarjota selkeä ja helppokäyttöinen käyttöliittymä, tuotannon työntekijöiden toiveiden mukaan. Tämä edellyttää tiedonkeruun ja visualisoinnin tehokasta toteutusta. Nopeasta tiedon saannista mobiilisovelluksen avulla on myös hyötyä ennakoivaan ylläpitoon ja nopeisiin reagointeihin häiriötilanteissa. Tämä tavoite liittyy suoraan mahdollisten kustannusten vähentämiseen ja kilpailukyvyn parantamiseen. Sovellus antaa myös arvokasta tietoa prosessista pitkällä aikavälillä, joten sen avulla pystytään analysoimaan ja optimoimaan tuotantoprosessia ajan mittaan.

Tavoitteisiin kuuluu myös arvioida mobiilisovelluksen käyttäjäkokemusta ja saada palautetta suoraan koksaamon työntekijöiltä. Käyttäjäkokemusten arviointi opinnäytetyön aikana auttaa varmistamaan, että sovellus täyttää käyttäjien tarpeet ja odotukset.

Raahen SSAB:n koksaamolla ei ole aikaisemmin ollut käytössä tällaista mobiilisovellusta, josta pystyy seuraamaan prosessin tilaa ja tärkeimpiä mittauksia. Aiemmin seuranta on perustunut koksaamon omalle web-palvelimelle kertyneestä tiedosta, jonka seuraaminen ja analysointi vie enemmän aikaa kuin mobiilisovelluksesta katsominen. Mobiilisovelluksen käyttöönoton myötä pyritään ratkaisemaan nämä haasteet ja tuomaan uutta arvoa koksaamon toimintaan. Reaaliaikaisen datan kerääminen ja nopea pääsy kriittisiin tietoihin auttavat parantamaan prosessin hallintaa sekä mahdollistavat nopean toiminnan poikkeamien ilmetessä.

Tämä opinnäytetyö tarjoaa erittäin hyvän tilaisuuden oppia ja soveltaa uutta osaamista mobiilisovelluskehityksen parissa erityisesti, koska siinä käytetään .NET MAUI-viitekehystä. Minulla ei ole aikaisempaa kokemusta .NET MAUIsta, joten tämä opinnäytetyö antaa hyvän mahdollisuuden oppia uusia taitoja sekä kohdata myös haasteita.

Projektin alussa minun täytyy perehtyä .NET MAUI-kehitysympäristöön, jotta sovelluksen tekeminen sujuu helpommin. Perusasioiden jälkeen siirrytään käyttöliittymän suunnitteluun ja toteutukseen sekä ohjelmoimaan sovelluksesta toiveiden mukaisen. .NET MAUIssa käytetään ohjelmointikielinä C# ja XAMLia, joista C# on minulle ennestään tuttu ohjelmointikieli, kun taas XAMLia en ole koskaan ennen käyttänyt.

Yksi merkittävistä oppimiskokemuksista tulee myös olemaan projektinhallinta ja aikataulutus. Työ täytyy jakaa eri vaiheisiin ja asettaa selkeät tavoitteet projektin etenemiselle. Sen edetessä saan kokemuksia varsinkin ongelmanratkaisusta, mikä on olennainen osa sovelluskehitystä, jossa tulee yleensä aina ongelmakohtia. Hyviä kokemuksia saan myös käyttöliittymän suunnittelemisesta ja toteuttamisesta, mikä on jäänyt opiskeluvuosieni aikana vähemmälle. Yhteistyö SSAB:n koksaamon kanssa opettaa minulle myös, kuinka tärkeää on viestiä ja kommunikoida tehokkaasti, jotta mobiilisovelluksesta saadaan toimiva ja tilaajan tarpeiden mukainen. Kaiken kaikkiaan tämä opinnäytetyö tarjoaa mahdollisuuden ymmärtää enemmän mobiilisovelluskehityksestä, eritoten .NET MAUI-viitekehyksen käyttämisestä järjestelmäriippumattomissa mobiilisovelluksissa.

Opinnäytetyöni aihe on hyvin ajankohtainen, sillä se vastaa nykyajan tarpeita monilla eri aloilla, erityisesti kuitenkin teollisuudessa. Aiheellani on myös suora vaikutus SSAB:n koksaamon toimintaan, sillä se mahdollistaa häiriötilanteiden havaitsemisen ja niihin reagoinnin, joka voi estää mahdollisia tuotantokatkoksia, millä taas voidaan säästää merkittäviä taloudellisia resursseja. Näin ollen opinnäytetyöni hyödyttää parantamalla prosessin tehokkuutta ja vähentämällä tuotantokustannuksia. Mobiilisovelluksen kehittäminen koksaamolle voi myös hyödyttää Raahen SSAB:n muita osastoja, jos se todetaan hyväksi ja toimivaksi seurantatyökaluksi, jolloin sitä pystyy jatkokehittämään soveltuvaksi muille osastoille.

1.3 Mobiilisovelluksen tekniikka

Mobiilisovellus tehdään Microsoft Visual Studio Enterprisellä käyttäen .NET MAUI viitekehystä, koska mobiilisovelluksesta tulee olla toimiva sovellus niin iOS- kuin Android -puhelimille. Visual Studio Enterprise valitaan alustaksi, koska SSAB:ltä löytyy lisenssit Microsoftille jo valmiiksi. Sovellukseen tulee esimerkiksi kuvaaja patterin ohjausmallin perustiedoista ja purkuaikataulu. Näiden tarkoituksena on helpottaa tietojen keräämistä ja raportointia, mikä parantaa päätöksentekoa, tehokkuutta sekä prosessinseurantaa.

Sovelluksen avulla saadaan myös laajempi yleiskuva prosessista ja mittauksista, mikä voi auttaa pitkällä aikavälillä optimoinnissa ja suunnittelussa. Nämä edut voivat myös helpottaa merkittävästi koksintuotantoprosessia.

Aiheen ajankohtaisuudessa mobiiliteknologian nopea kehitys ja sen yleistyminen teollisuuden eri osa-alueilla tekee mobiilisovelluksen roolista merkittävän. Se korostuu erityisesti teollisuusprosessien optimoinnissa ja tuotannon valvonnassa.

1.4 Opinnäytetyön vaiheet

Opinnäytetyöni aikataulutus koostuu useista eri vaiheista, mukaan lukien tutkimuksen suunnittelu, mobiilisovelluksen suunnittelu ja kehitys sekä käyttäjäkokemusten arviointi. Mobiilisovelluksen suunnittelu aloitettiin jo ennen varsinaista opinnäytetyön aloittamista harjoittelupaikassani SSAB:lla ohjaajani kanssa. Suunnittelimme millä menetelmillä mobiilisovellus toteutetaan, jotta se vastaisi koksaamon tuotannon tarpeita. Mobiilisovelluksen pitäisi soveltua niin Android- kuin iOS-laitteille, koska eri työntekijöillä voi olla käytössä molempien käyttöjärjestelmien puhelimia. Tämän pohjalta päädyimme ratkaisuun, että käyttäisin työn alustana Microsoft Visual Studio Enterpriseä ja .NET MAUI -viitekehystä.

Suunnittelimme valmiiksi, mitä kuvaajia ja taulukoita sovellukseen ainakin tulisi, esimerkiksi aikaisemmin mainitut lämmönohjausmalli ja purkuaikataulu. Tarkkaa suunnitelmaa tästä ei kuitenkaan tehty, koska sovelluksen käyttäjien halutaan itse pääsevän vaikuttamaan, mitä sovelluksessa tulee näkymään, jotta sovelluksesta tulee juuri toiveiden mukainen.

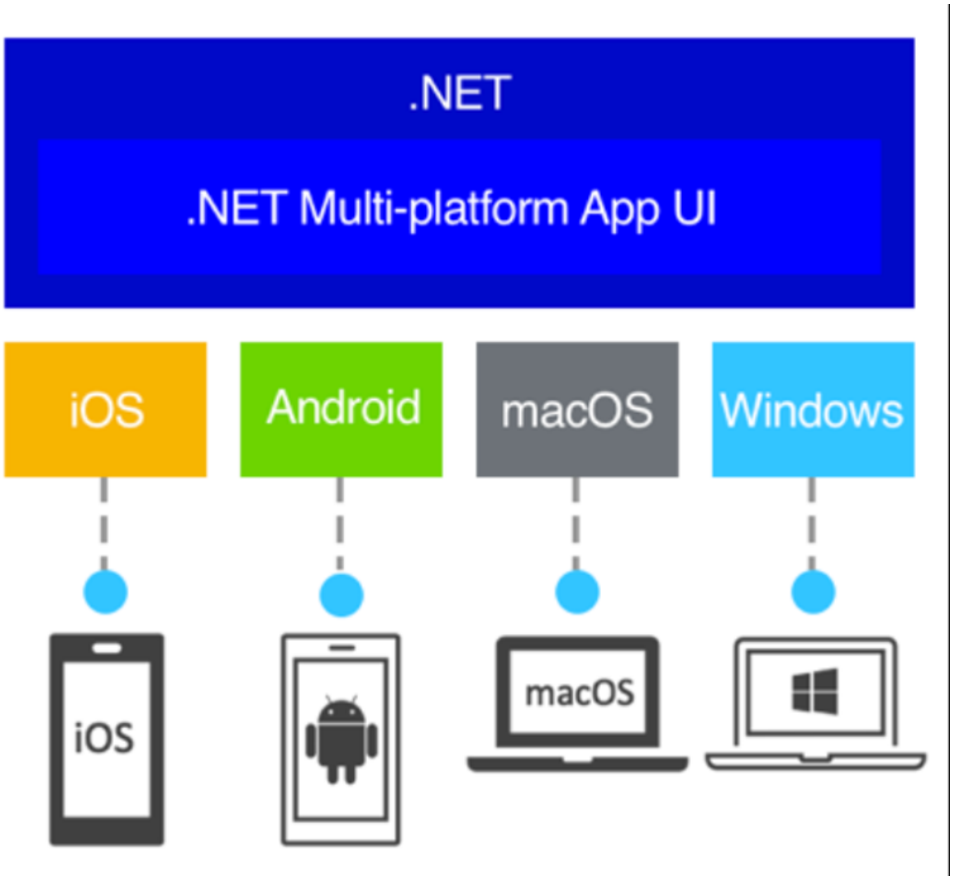
2 MULTI-PLATFORM APP USER INTERFACE

.NET MAUI on Android-, iOS-, macOS- ja Windows-käyttöjärjestelmille tarkoitettu kehys, joka on kehitetty Xamarin.Forms:sta. Sen koko järjestelmä on päivitetty ja nykyaikaistettu, jotta kehittäminen olisi entistäkin helpompaa. .NET MAUI on julkaistu toukokuussa 2022. (Pluszczewska 2023.)

2.1 Monialustainen viitekehys

.NET MAUI eli Multi-platform App User Interface on suunniteltu helpottamaan sovellusten kehittämistä useille eri alustoille yhdellä ja samalla koodipohjalla. Se on avoimen lähdekoodin versio, joka on laajennettu mobiililaitteista työpöytäversioihin ja sen käyttöliittymäsäätimet on rakennettu alusta alkaen uudelleen suorituskyvyn ja laajennettavuuden parantamiseksi. (Microsoft 2023a.)

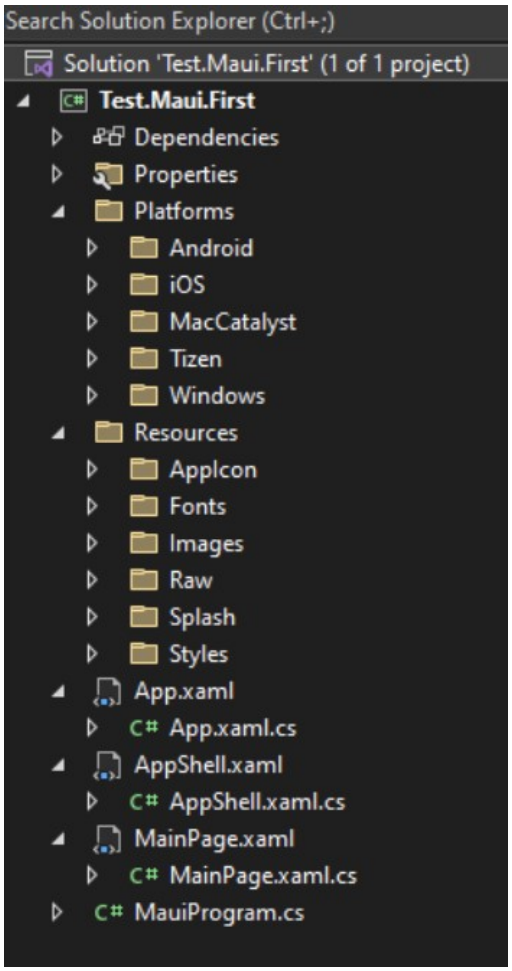
Yksi .NET MAUIn tärkeimmistä tavoitteista on mahdollistaa mahdollisimman suuren osan sovelluslogiikasta ja käyttöliittymäasettelusta yhdessä koodikannassa. Se tarjoaa myös kokoelman ohjauskomponentteja, joita voidaan käyttää esimerkiksi datan näyttämiseen, toimintojen aloittamiseen, toiminnan osoittamiseen, kokoelmien näyttämiseen ja datan valintaan. (sama.) Kuvassa 1 havainnollistetaan .NET MAUIa monialustaisena sovelluskehitysympäristönä.



KUVA 1. .NET MAUI (Microsoft 2023a)

.NET tarkoittaa Microsoftin kehittämää ohjelmistokomponenttikirjastoa ja se on ilmainen avoimen lähdekoodin monialustainen kehys nykyaikaisten sovellusten ja tehokkaiden pilvipalvelujen rakentamiseen (Microsoft 2023c). Ohjauskomponenttikokoelman lisäksi se myös tarjoaa esimerkiksi yksityiskohtaisen sivujen suunnitellun asetteluohjelman, useita sivutyyppejä navigoinnin luomiseksi ja tuen datan sitomiseen, mikä mahdollistaa tyylikkäämmät ja ylläpidettävämmät kehitysmallit (Microsoft 2023a).

.NET MAUI-projektin rakenteeseen kuuluu muun muassa alusta eli ”Platforms” ja resurssit eli ”Resources” kansiot. Alusta kansio sisältää alikansiot Android-, iOS-, MacCatalyst-, Tizen- ja Windows -käyttöliittymille. (Kuva 2.) Nämä tiedostot käyttävät MauiProgram-luokkaa sovelluksen suorittamiseen, joka suoritetaan sovelluksen käynnistyessä. MauiProgram-luokka määrittää sovelluksen visuaalisen ulkoasun. Resources-kansio sisältää sovelluksessa käytetyt resurssitiedostot, kuten esimerkiksi fontit, tyylit ja kuvat. (Danyil 2022.)



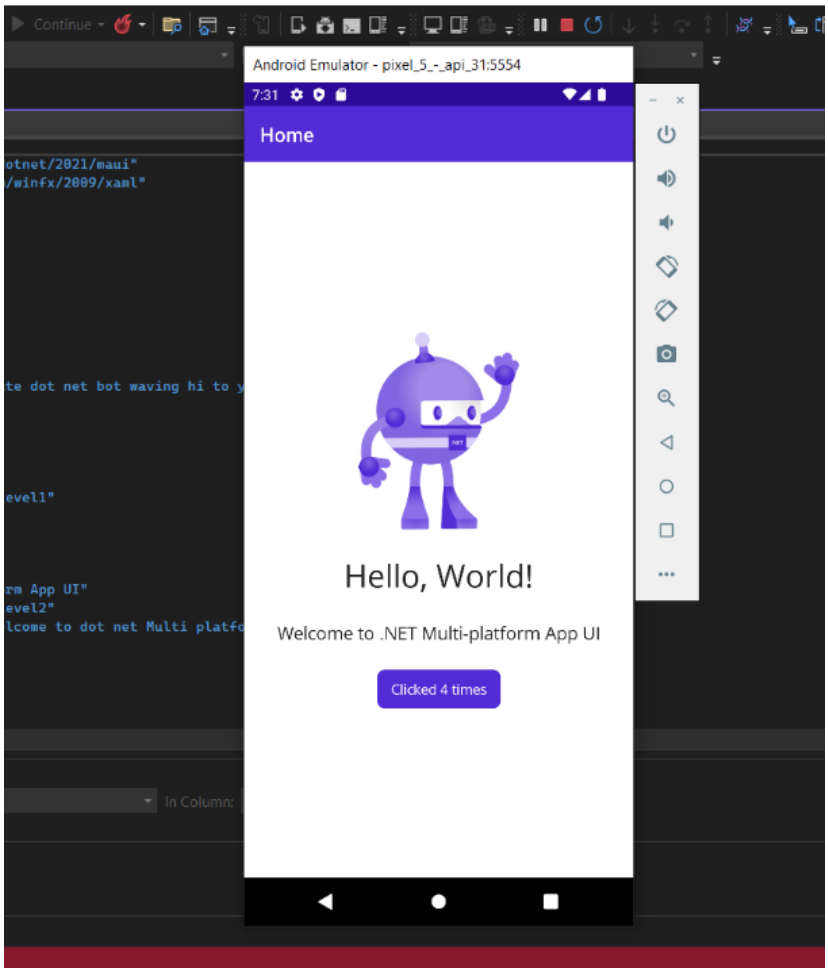
KUVA 2. .NET MAUI-projektin rakenne (Danyil 2022)

App.xaml määrittää resurssit, jotka ovat yhteisiä koko sovellukselle, ja App.xaml.cs on C#-koodi-tiedosto, josta sovellus käynnistyy. AppShell.xaml määrittelee monisivuisen sovelluksen yleisen visuaalisen käyttöliittymän. AppShell.xaml.cs liittyy edellä mainittuun tiedostoon ja määrittää siihen liittyvän ohjelmalogiikan. MainPage.xaml on pääsivun visuaalinen käyttöliittymätiedosto ja siihen kuuluva MainPage.xaml.cs tiedosto sisältää pääsivun logiikan. (Danyil 2022.)

App-luokka käyttää AppShell-luokkaa AppShell.xaml.cs ja siihen liittyvästä AppShell.xaml-tiedostosta määrittämään, kuinka sivut rakennetaan sovelluksessa. AppShell-luokan avulla voidaan luoda monisivuinen sovellus ja määrittää sovelluksen pääsivu, jota MainPage-luokka yleisesti edustaa. (sama.)

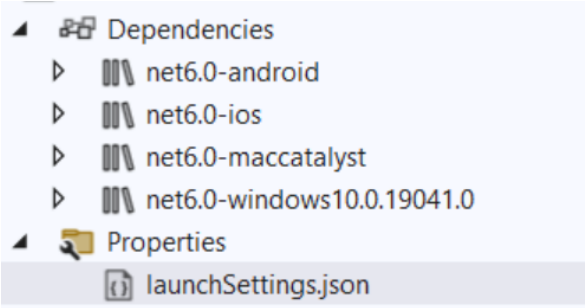
Sovelluksen käynnistäminen kehityksen aikana eri käyttöliittymissä tapahtuu käyttämällä älypuhe-linemulaattoria (kuva 3). Emulaattori valitaan käynnistysasetuksista, jolloin Framework-kentän tu-

lee olla valitun käyttöliittymän mukainen, kuten esimerkiksi Androidissa .net6.0-android. Emulaattori tulee kuitenkin luoda ensin, jolloin sen mallin voi itse valita. Valittu malli vaikuttaa älypuhelinemulaattorin kokoon. Sen jälkeen kaikki tiedostot ladataan, käynnistetään emulaattori ja lopuksi ajetaan sovellus, jolloin se avautuu valittuun emulaattoriin. (Danyil 2022.)



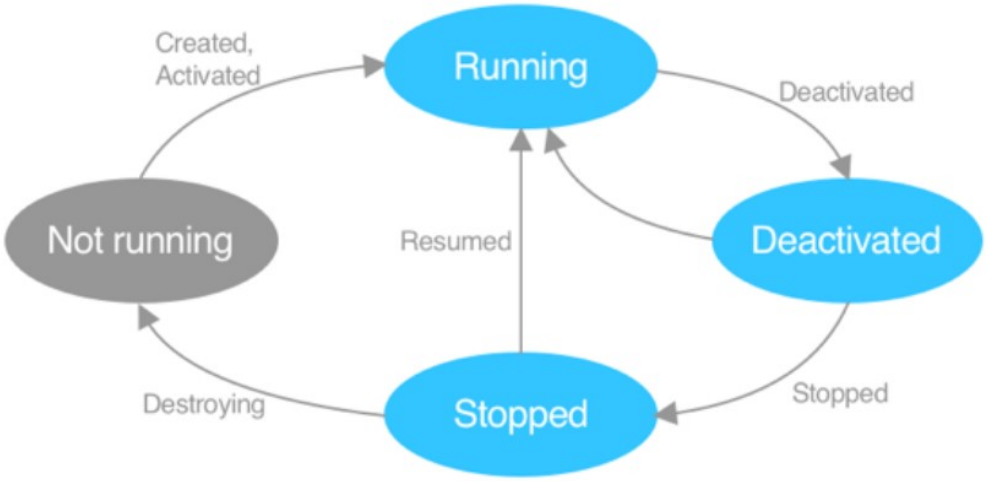
KUVA 3. Android-emulaattori (Danyil 2022)

MAUI-sovelluksen riippuvuudet eli ”Dependencies” voivat olla SDK:ita, NuGet-paketteja, kolmannen osapuolen kirjastoja ja mukautettuja komponentteja. Ominaisuudet eli Properties -kansiossa voi olla esimerkiksi launchSettings.json-tiedosto, jota käytetään sovelluksen käynnistyksen ja virheenkorjauksen asetusten määrittämiseen. (Sharma 2023.) Kuvassa 4 nähdään esimerkki launchSettings.json tiedostosta, jossa voidaan määrittää profiileja, ympäristömuuttujia ja argumentteja.



KUVA 4. Ominaisuudet kansion launchSettings.json-tiedosto (Sharma 2023)

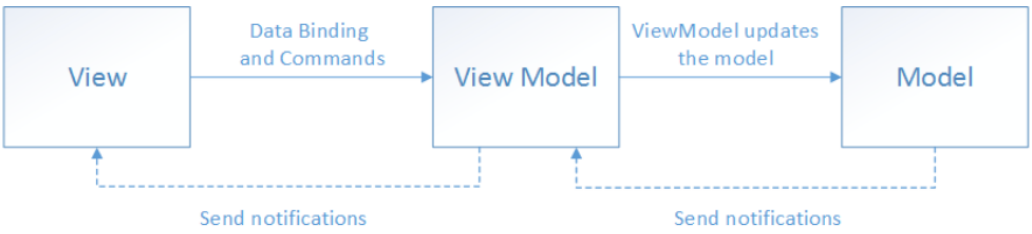
.NET MAUI-sovelluksissa on yleensä neljä eri suoritustilaa, jotka ovat ei käynnissä, käynnissä deaktivoitu ja pysäytetty (kuva 5). Sovelluksen suoritustila riippuu sovelluksen historiasta. Esimerkiksi, kun sovellus asennetaan ensimmäistä kertaa, tila on ei käynnissä. Kun taas sovellus käynnistetään, se on käynnissä olevassa tilassa. Jos eri sovellusikkuna avataan, siirtyy se deaktivoituun tilaan ja sovellus on toimintakyvytön. Kun sovellus vaihdetaan toiseen tai palataan aloitusnäyttöön, deaktivoitu- ja pysäytetty -tapahtumat tulevat voimaan ja sovellus pysäytetään. (Microsoft 2023d.)



KUVA 5. Sovelluksen elinkaari (Microsoft 2023d)

.NET MAUI-sovelluksen arkkitehtuurina käytetään MVVM-mallia (Microsoft 2023f). MVVM-lyhenne tarkoittaa Model-View-ViewModel suunnittelumallia, joka on jäsennelty erottamaan ohjelmalogiikka ja käyttöliittymänohjaimet. MVVM auttaa järjestämään koodia ja jakamaan ohjaimia moduuleiksi, mikä tekee koodin kehittämisestä, päivittämisestä ja uudelleenkäytöstä yksinkertaisempaa ja nopeampaa. (TechTarget 2019.) MVVM:ssä Model ja ViewModel ovat kokonaan koodilla kirjoitettuja

luokkia, jossa ViewModel viittaa XAML-tiedostoon (Microsoft 2023f). Alla oleva kuvassa mallinnetaan näiden kolmen ydinkomponentin välisiä suhteita ja miten ne toimivat vuorovaikutuksessa toisiinsa (kuva 6).

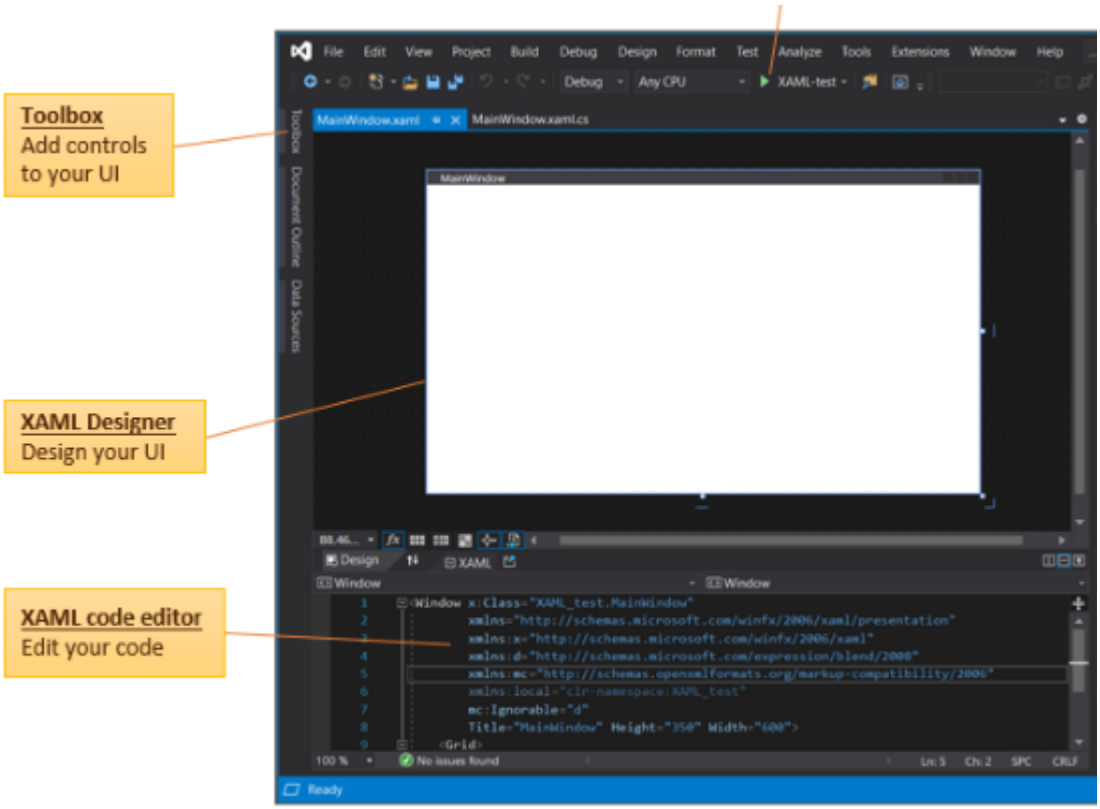


KUVA 6. MVVM-malli (Microsoft 2022a)

2.2 XAML

XAML, joka on lyhenne sanoista ”Extensible Application Markup Language” on XML- pohjainen ohjelmointikieli. XAMLia käytetään pääasiassa määrittämään sivujen visuaalinen sisältö ja se toimii yhdessä C#-koodin takana olevan tiedoston kanssa. Yhdessä nämä tiedostot muodostavat uuden luokan määritelmän, joka sisältää aliluokat ja ominaisuuden alustamisen. (Microsoft 2023b.)

XAML-tiedostossa pystyy määrittämään näkymät, näkymien asettelun, värit, fontit, animaatiot ja paljon muuta. Jokainen käyttöliittymäelementti määritellään XML- elementtinä, joka voi sisältää joukon attribuutteja ja arvoja. XAMLilla voi rakentaa monimutkaisiakin näkymiä liittämällä yhteen erilaisia elementtejä, kuten painikkeita, tekstikenttiä, kuvia, luetteloita, kuvaajia ja taulukoita. Näiden elementtien sijoittelua näytöille pystyy myös määrittelemään käyttämällä asettelukontrollereita. XAML mahdollistaa myös tyylien, resurssien ja mallien käytön, mikä tekee sovelluksen ulkoasun ylläpidosta ja kustomoinnista joustavaa. Sillä on myös mahdollista määrittää yksi tyyli ja käyttää sitä useissa paikoissa, mikä auttaa ylläpitämään yhteensopivaa ulkoasua. (sama.) XAMLissa on myös mahdollista rakentaa käyttöliittymää sijoittelemalla komponentteja suoraan näyttöön (kuva 7).



KUVA 7. XAML Microsoft Visual Studiossa (Microsoft 2022b)

XAML on olennainen osa .NET MAUI-sovelluskehitystä, joka mahdollistaa tehokkaan visuaalisen suunnittelun ja erottaa sen selkeästi sovelluslogiikasta. Sen lähestymistapa tekee käyttöliittymän rakentamisesta joustavaa sekä se tarjoaa runsaasti mahdollisuuksia luoda houkuttelevia ja käyttäjäystävällisiä mobiilisovelluksia. (sama.)

XAMLissa objektielementin syntaksit alkavat aina avauskulmasululla. Tätä seuraa tyypin nimi, johon halutaan luoda instanssi. Tämän lisäksi voidaan lisätä attribuutteja objektielementtiin. Objektielementti viimeistellään sululla ja sulkevalla kulmasululla peräkkäin. (Tizen 2023.) Kuvassa 8 nähdään esimerkki, kuinka instanssina käytetty yksinkertainen TextLabel-komponentti luodaan.

XML Kopio

```xml
<TextLabel Text="TextLabel"/>
```

KUVA 8. Etiketin luominen (Tizen 2023)

Attribuuttisyntaksi nimeää ominaisuuden aina tietyn attribuutin arvoksi ja sen jälkeen yhtäsuuruusmerkillä määritetään tietyt tehtävät. Attribuutin arvo ilmoitetaan aina merkkijonona lainausmerkkien sisällä. (Tizen 2023.) Tällä koodilla luodaan TextLabel-komponentti, jolla on sinisellä taustalla ja punaisella kirjoitettu ”TextLabel” (kuva 9).

XML Kopio

```xml
<TextLabel TextColor="Red" BackgroundColor="Blue" Text="TextLabel" />
```

KUVA 9. Värien luomista etikettiin (Tizen 2023)

Ominaisuuselementin aloitustunnisteen syntaksi on ”<typeName.propertyName>”. Sen sisältö on kyseisen ominaisuuden arvon ottavan tyypin objektielementti. Kun sisältö on määritelty, ominaisuuselementti suljetaan lisäämällä sulku rivin alkuun. (Tizen 2023.) Tässä esimerkissä on samat ominaisuudet kuin edellisessä attribuuttisyntaksissa, mutta siinä on käytetty ominaisuuselementtisyntaksia kaikille ominaisuuksille (kuva 10).

XML Copy

```xml
<TextLabel>
  <TextLabel.TextColor>
    Red
  </TextLabel.TextColor>
  <TextLabel.BackgroundColor>
    Blue
  </TextLabel.BackgroundColor>
  <TextLabel.Text>
    TextLabel
  </TextLabel.Text>
</TextLabel>
```

KUVA 10. Värien käyttämistä ominaisuussyntaksilla (Tizen 2023)

2.3 Käyttöliittymäkomponentit

Käyttöliittymä eli ”user interface/UI” on se ohjelmiston tai laitteen näkyvä osa, jota käyttäjä käyttää (Hurja 2021.). Erilaisia käyttöliittymä tyyppejä voivat olla esimerkiksi graafinen käyttöliittymä, komentoriviliittymä, valikkopohjainen käyttöliittymä, kosketuskäyttöliittymä, äänikäyttöliittymä, lomakepohjainen käyttöliittymä tai mobiilikäyttöliittymä (Churchville 2021).

Käyttöliittymäsuunnittelussa tärkeintä on, että verkkosivusto, mobiilisovellus tai jokin muu ohjelmisto on suunniteltu käyttäjien tarpeisiin. Hyvä käyttöliittymä tulisi olla selkeä ja looginen, vuorovaikutuksellinen, mukautuva ja saavutettava. (Hurja 2021.)

.NET MAUIssa on käytössä paljon eri käyttöliittymäkomponentteja, kuten asettelusäätimiä, painikkeita, datanäyttöjä, päivämäärä näyttöjä ja valintoja, valikoita, valintasäätimiä, navigointiohjaimia, valintaikkunoita, käyttäjätietojen ohjausobjekteja, asiakirjoja, syöttösäätimiä sekä ääni- ja kuvasisältökomponentteja (kuva 11). Asettelusäätimiä käytetään alielementtien koon, mittojen, sijainnin ja järjestelyn hallintaan. Painikkeet ovat yksi peruskäyttöliittymän ohjaimista, joilla suoritetaan tehtäviä, kun niitä napsautetaan. Microsoftin lisäksi käyttöliittymäkomponentteja tarjoavat kolmannen osapuolen kirjastot. Kolmannen osapuolen kirjastot löytyvät Microsoft Visual Studion sisäisestä NuGet Package Manager työkalusta, josta ne voidaan ladata projektiin. Kun kirjasto on ladattu, täytyy se myös lisätä C#-luokkaan käyttäen using-lausetta. (Microsoft 2023e.)

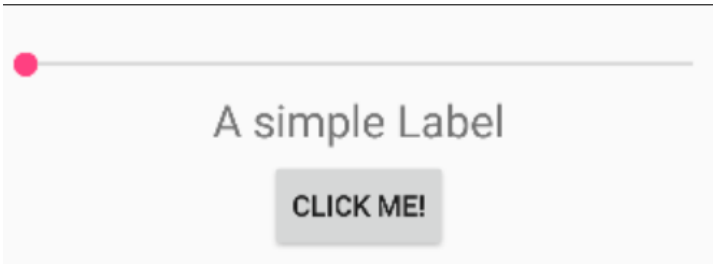
KUVA 11. .NET MAUIn tarjoamat komponentit kategorioissaan sivut, asettelusäätimet ja näkymät (Microsoft 2023g)

XAML-tiedostossa voidaan käyttää asettelusäätimenä Stacklayoutia, joka tarkoittaa elementtien sijoittelua yhdelle riville, jotka voidaan suunnata pysty- tai vaakasuuntaan. VerticalOptions määrittelee komponenttien sijainnin vaakasuunnassa ja HorizontalOptions pystysuunnassa. Stacklayoutin sisään on sijoitettu liukusäädin, tekstikenttä ja painike, jotka näkyvät asettelun mukaan päällekkäin. (Kuva 12.)

```xaml
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="XamlSamples.XamlPlusCodePage"
             Title="XAML + Code Page">
    <StackLayout>
        <Slider VerticalOptions="Center" />
        <Label Text="A simple Label"
               FontSize="18"
               HorizontalOptions="Center"
               VerticalOptions="Center" />
        <Button Text="Click Me!"
                HorizontalOptions="Center"
                VerticalOptions="Center" />
    </StackLayout>
</ContentPage>
```

KUVA 12. XAML asettelusäätimet (Microsoft 2023b)

Edellä kuvattu XAML-koodi muodostaa seuraavanlaisen käyttöliittymän (kuva 13). Label:lla muodostetaan yksirivinen tekstielementti, jossa sen tulostukseksi on kirjoitettu ”A simple Label” sekä tekstin fonttikooksi on asetettu 18. Button tarkoittaa painiketta, jonka toiminnot voidaan määrittää XAML-tiedostoon liittyvässä C# -koodin takana olevassa tiedostossa. (Microsoft 2023b.)

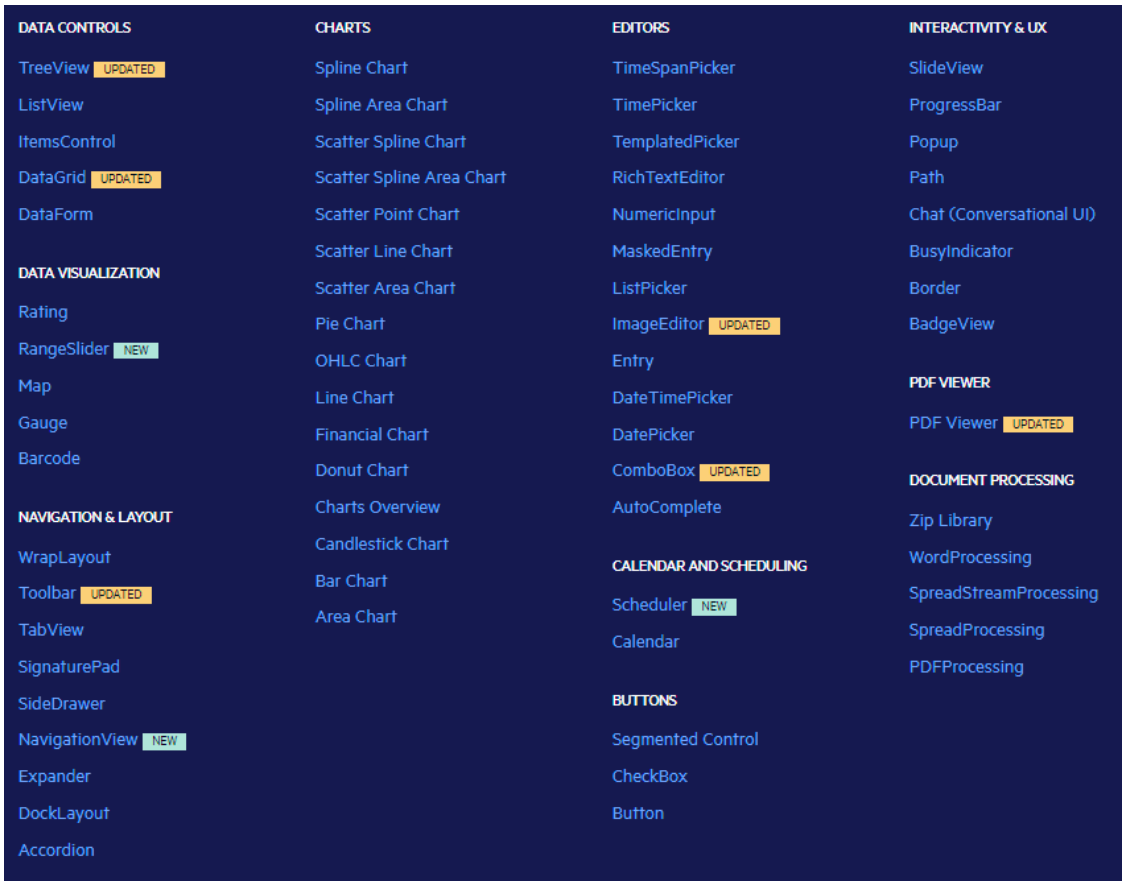


KUVA 13. XAML-koodin tulostus (Microsoft 2023b)

2.4 Telerik

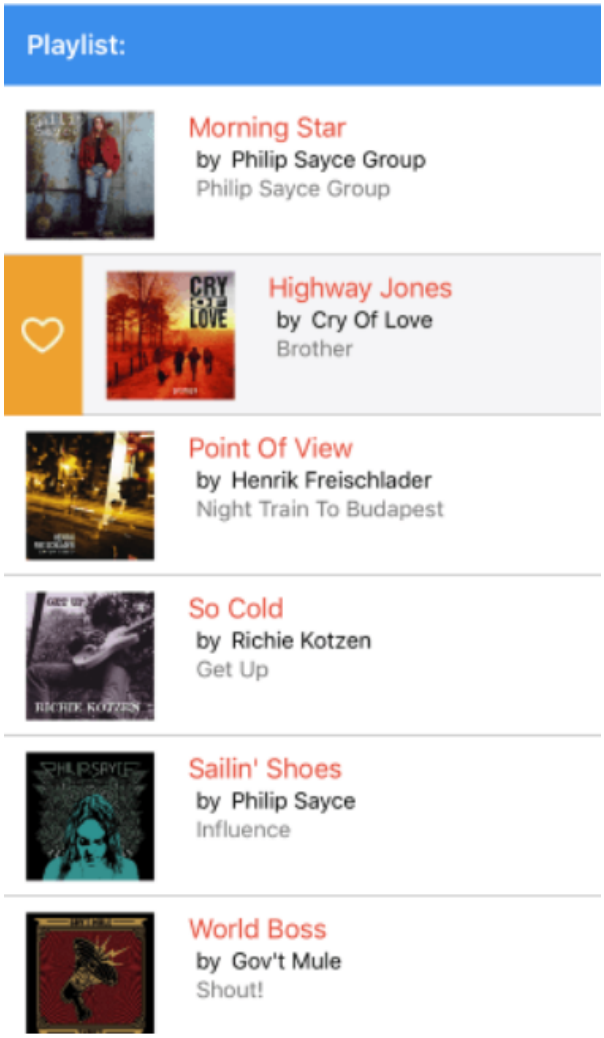
Telerik UI on yksi .NET MAUIin saatavista käyttöliittymäkirjastoista markkinoiden laajin ja se tarjoaa ohjaimia modernin näköiseen mobiili- ja työpöytäsovellusten rakentamiseen Android, iOS-, macOS- ja Windows-alustoille yhdestä koodikannasta (Telerik 2023a). Telerik UI for .NET MAUI tarjoaa yli 60 ohjainta täyttämään kaikki järjestelmäriippumattoman sovelluksen vaatimukset tietojen käsittelyssä, suorituskyvyssä, käyttökokemuksessa ja suunnittelussa (Telerik 2023b). Telerik on

kolmannen osapuolen kirjasto ja se helpottaa sekä nopeuttaa koodausta, koska se tarjoaa toiminnallisuutta, jotta koodaajien ei tarvitse itse koodata komponentteja (Telerik 2023a). Kuvassa 14 on listattu Telerik-kirjaston tarjoamat ohjaimet .NET MAUIlle.



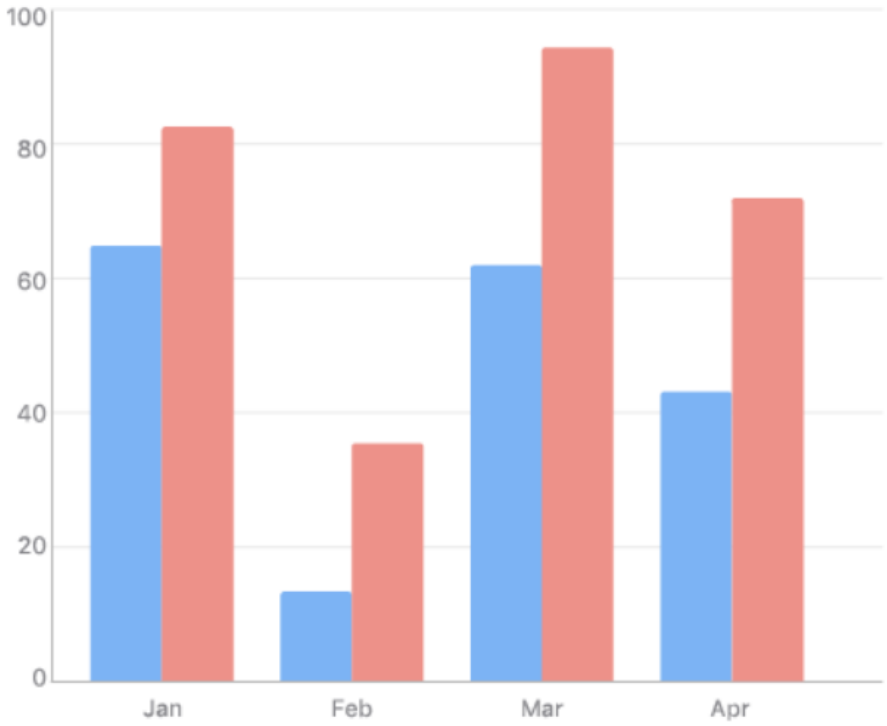
KUVA 14. Lista Telerik-kirjaston ohjaimista (Telerik 2023b)

Telerikin tarjoamat ohjaimet voivat olla esimerkiksi tietojen hallintaan- ja visualisointiin, navigointiin ja asetteluun, kaavioihin ja editoreihin liittyviä ohjaimia (Telerik 2023b). Tietojen hallinnasta ListView on luettelokomponentti, joka sitoo tiedot yhteen luetteloon (kuva 15). Näitä tietoja voidaan suodattaa, ryhmitellä ja lajitella luettelossa (Telerik 2023c).



KUVA 15. Listanäkymä (Telerik 2023c)

Pylväskaavio on yksi yleisimmistä kaavioista ja se sopii hyvin erityyppisten tietojen vertailuun yksinkertaisuutensa vuoksi. Pylväskaavion voi visualisoida näyttämään tiedot joko vaaka- tai pystysuunnassa. (Telerik 2023d.) Seuraavassa kuvassa 16 nähdään, että pylväskaavio on asetettu näyttämään tiedot pystysuunnassa.



KUVA 16. Pylväskaavio, jossa tiedot näytetään pystysuunnassa (Telerik 2023d)

Numeerinen akselikaaviotyyppi on välttämätön osa suorakulmaista koordinaatiojärjestelmää. Kaaviotyyppi piirtää liittyvät datapisteet käyttämällä kunkin pisteen numeerista arvoa, joka on annettu akselille. (Telerik 2023e.) Kuvassa 17 nähdään, kuinka tämä toteutetaan XAML-koodissa.

XAML

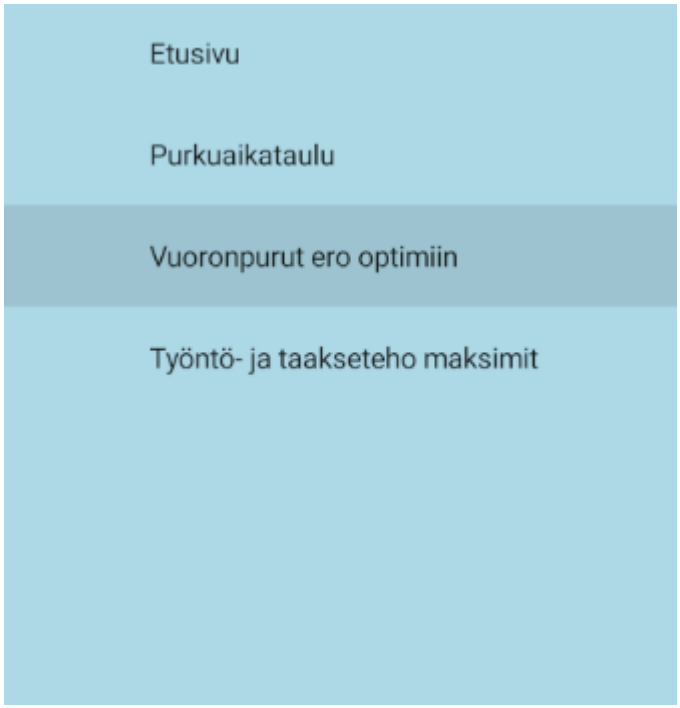
```
<telerik:NumericalAxis LabelFormat="C"
                       MajorStep="0.5"
                       Minimum="-1"
                       Maximum="1" />
```

KUVA 17. Esimerkki XAML-koodista, kuinka datapisteet kaaviolle asetetaan (Telerik 2023e)

3 MOBIILISOVELLUKSEN TOTEUTUS

3.1 Sovelluksen toiminnot

Sovellukseen rakennettiin neljä eri sivua säätiedoille, purkuaikataululle, vuoronpurut ero optimiin sekä työntö- ja taakseteho maksimeille. Näihin sivuihin tiedot haettiin SSAB:n omasta SQL Server tietokannasta. .NET MAUI-sovelluksen lisäksi täytyi luoda erillinen projekti ohjelmistorajapinnalle, joka mahdollisti tietokannan tietojen välittämisen Android-laitteille. Käytin REST-mallia tämän toteuttamiseen, jonka avulla sain JSON-muotoiset tiedot haettua GET-pyynnöillä sovellukseeni. Kuvassa 18 nähdään sovelluksen navigointipalkki.



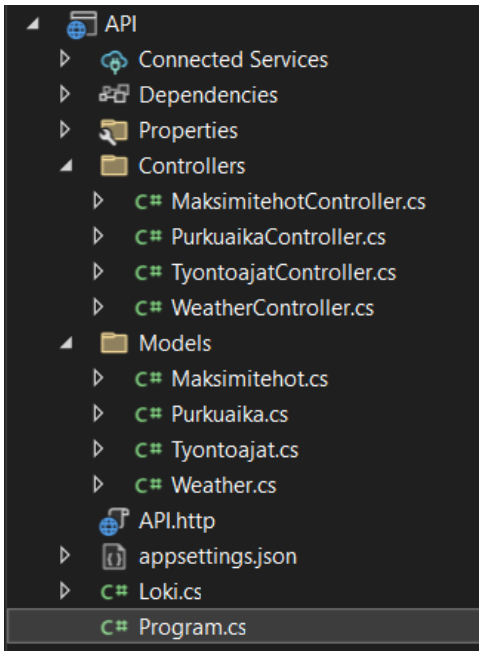
KUVA 18. Sovelluksen navigointipalkki

Purkuaikataulu sisältää taulukon, jossa näkyy uunien numerot 1-70 siinä järjestyksessä, mikä on purkuvuorossa seuraavana. Lisäksi taulukossa näkyy uunien panostus- ja purkuajat. Vuoronpurut ero optimiin sivulla taulukossa näkyy kellonajasta riippuen joko aamuvuoron eli kello 6:00-18:00 merkatut mittaukset tai ilta/yövuoron eli 18:00-6:00 mittaukset. Tähän taulukkoon on laitettu näkyviin uunien numerot, uunin työntöaika sekä ero optimiin minuutteina. Ero optimiin kohta tarkoittaa uunin tavoite purkuajan eroa optimiin minuutteina. Eli, jos luku on negatiivinen, uunin purku on

tapahtunut edellä optimaalisesta aikataulusta. Jos luku on positiivinen, uunin purku on jäljessä optimaalista aikataulua. Maksimitehot sivulla on pylväskaavio, joka päivittyy uunin numeron mukaan. Kaavioon tuli työntö- ja taakseteho maksimeiden viisi viimeisintä mittausta, jotka tarkoittavat uunin purku- tai työntövaiheessa mitattuja maksimi sähkövirtoja ampeereina.

3.2 Web-rajapinta

ASP.NET CORE Web API projektin arkkitehtuurina käytettiin MVC-mallia. MVC koostuu sanoista Model, View, Controller. Model-tietomalli kuvaa sovelluksen tietoja eli tässä tapauksessa käytettyjä tietokannan taulun sarakkeita, kuten esimerkiksi uunit ja aikataulut. Controller-ohjain vastaa syötteiden käsittelystä ja ohjaa tietovirtoja. Projektissa käytettiin controllereita tekemään HTTP GET-pyyntöjä tietokantaan ja palauttamaan tiedot JSON-muodossa. View-näkymiä ei tähän projektiin tullut, koska näkymät ovat tehty .NET MAUI-projektissa. Kuvassa 19 nähdään API-projektin rakenne Models- ja Controllers-luokkineen.



KUVA 19. ASP.NET Core WEB API-projekti

Models-luokat sisältävät tietokannan taulun käytetyt sarakkeet. Kuvassa 20 nähdään, että Purkuaika-sivulle on haettu tiedot STATE-taulusta ja sarakkeina on käytetty OVEN, TALKA ja TYLKA, jotka tarkoittavat tietyn uunin numeroa, sen panostusaikaa ja purkuaikaa.

```
1   using System.ComponentModel.DataAnnotations;
2   using System.ComponentModel.DataAnnotations.Schema;
3
4   namespace API.Models
5   {
6       [Table("STATE")]
        4 references
7       public class Purkuaika
8       {
9           [Key]
            1 reference
10          public decimal OVEN { get; set; }
11
            1 reference
12          public DateTime TALKA { get; set; }
13
            1 reference
14          public DateTime TYLKA { get; set; }
15      }
16  }
17
```

KUVA 19. Purkuaika-luokka

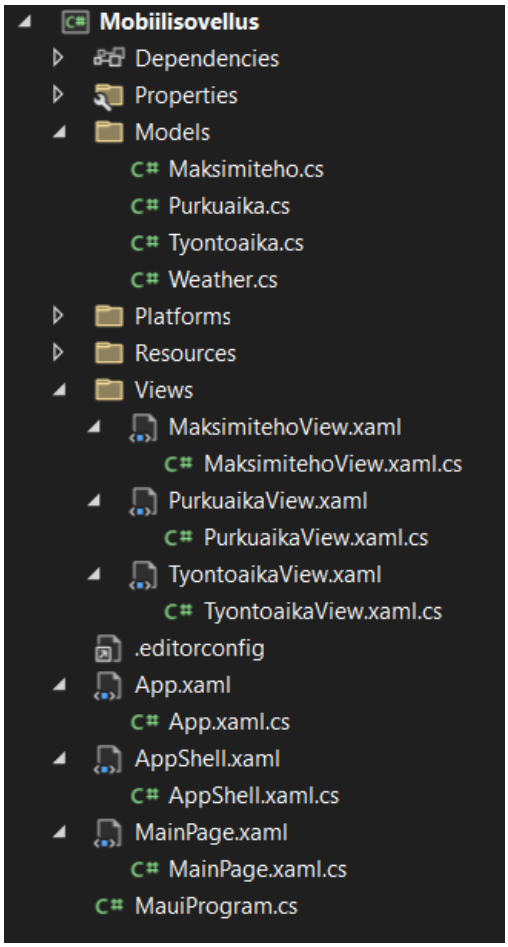
Controllers-luokat käsittelevät HTTP -pyyntöjä, tässä tapauksessa kuitenkin vain GET -pyyntöjä tietyn polun kautta eli [Route(”[controller]”)], jossa controller on tässä luokassa Purkuaika. HttpGet-metodi hakee purkuaikatiedot tietokannasta SQL-kyselyllä, täyttää tulokset DataTable-olioon ja lopuksi palauttaa JSON-muodossa olevat purkuaikatiedot. (Kuva 20.)

```
9   namespace API.Controllers
10  {
11      [Route("[controller]")]
12      [ApiController]
        1 reference
13      public class PurkuaikaController : ControllerBase
14      {
15
16          public readonly IConfiguration _configuration;
17
            0 references
18          public PurkuaikaController(IConfiguration configuration)
19          {
20              _configuration = configuration;
21          }
22
23          [HttpGet(Name = "GetPurkuaika")]
24
            0 references
25          public string Purkuaikatiedot()
26          {
27              Loki lw = new Loki();
28              List<Purkuaika> purkuajat = new List<Purkuaika>();
29
30              try
31              {
32                  SqlConnection conn = new SqlConnection(_configuration.GetConnectionString("KOKSTI").ToString());
33                  SqlDataAdapter dr = new SqlDataAdapter("SELECT OVEN as Uuni, TALKA as Panostusaika, TYLKA as Purkuaika FROM STATE
34                  DataTable dt = new DataTable();
35                  dr.Fill(dt);
36
37                  if (dt.Rows.Count > 0)
38                  {
39                      for (int i = 0; i < dt.Rows.Count; i++)
40                      {
41                          Purkuaika purkuaika = new Purkuaika();
42                          purkuaika.OVEN = Convert.ToInt32(dt.Rows[i]["Uuni"]);
43                          purkuaika.TALKA = Convert.ToDateTime(dt.Rows[i]["Panostusaika"]);
44                          purkuaika.TYLKA = Convert.ToDateTime(dt.Rows[i]["Purkuaika"]);
45                          purkuajat.Add(purkuaika);
46                      }
47                  }
48              }
```

KUVA 20. PurkuaikaController-luokka

3.3 Sovelluksen rakenne

.NET MAUI-projektin arkkitehtuurina käytettiin MVVM-mallia. MVVM-mallissa kirjaimet tarkoittavat Model, View, ViewModel ja ne erottavat sovelluksen datamallin sekä käyttöliittymän toisistaan. Model vastaa sovelluksen datamallista eli Models-luokista, jotka ovat luotu samalla periaatteella kuin API-projektissa. View on käyttöliittymäkomponentti, joka vastaa sovelluksen näkymästä. Views-kansio sisältää ContentPage-sivuja jokaiselle näkymälle, joka luo xaml- ja cs-tiedostot. XAML-tiedostossa määritellään sivun näkymä ja cs-tiedostossa määritellään tämän sivun toiminnot. ViewModel yhdistää Modelin ja Viewin. Se sisältää logiikan, joka hallitsee käyttöliittymän tilaa Data Bindingin avulla. Eli, kun komponentti yhdistetään Data Bindingia käyttämällä koodiin ja ViewModelin tila muuttuu, näkymä päivittyy automaattisesti. Kuvassa 21 on .NET MAUI-projektin tiedostorakenne.



KUVA 21. .NET MAUI-projektin rakenne

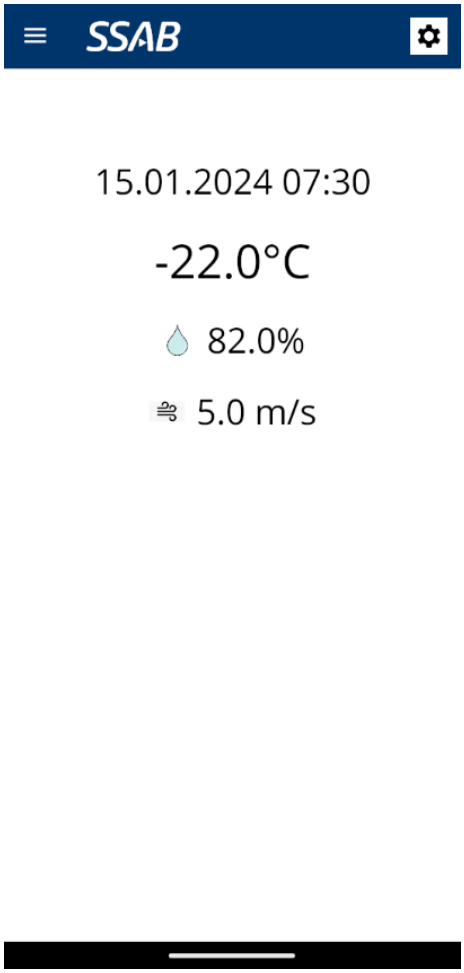
PurkuaikaView.xaml.cs-luokassa määritellään Purkuaikataulu näkymän toiminnot. JSON-muotoinen data haetaan REST-rajapinnasta, johon GET-pyyntö tehdään. Rajapinnan vastaus luetaan ensin merkkijonona, jonka jälkeen se deserialisoidaan C#-olioon. Datalle alustetaan lista ja lopuksi se syötetään DataGrid-taulukkoon käyttämällä ItemsSource-ominaisuutta. (Kuva 22.)

```
8   public partial class PurkuaikaView : ContentPage
9   {
10      //Parillisten ja parittomien data näihin
        3 references
11      public List<Purkuaika> PurkuaikaData { get; set; }
12
13      [Obsolete]
        0 references
14      public PurkuaikaView()
15      {
16          InitializeComponent();
17
18          //Kutsutaan GetPurkuajat metodia joka minuutti, jotta taulukko päivittyy
19          Device.StartTimer(TimeSpan.FromMinutes(1), () =>
20          {
21              GetPurkuajat();
22              return true;
23          });
24          //Kutsutaan GetPurkuajat metodia ensimmäisen kerran sivun alustamiseksi
25          GetPurkuajat();
26      }
27
        2 references
28      private async void GetPurkuajat()
29      {
30          try
31          {
32              //HttpClient-olio, REST-pyyntöjä varten
33              var httpClient = new HttpClient();
34              //GET-pyyntö annettuun osoitteeseen
35              var response = await httpClient.GetAsync("http://146.81.18.20:8081/Purkuaika");
36
37              if (response.IsSuccessStatusCode)
38              {
39                  //Luetaan vastaus merkkijonona
40                  var data = await response.Content.ReadAsStringAsync();
41                  //Deserialisoidaan JSON-muotoinen data
42                  var purkuajat = JsonConvert.DeserializeObject<List<Purkuaika>>(data);
43
44                  //Alustetaan uudet listat
45                  PurkuaikaData = new List<Purkuaika>();
46
47                  foreach (var purkuaika in purkuajat)
48                  {
49
50                      //Muutetaan vahingossa jääneestä decimal muodosta int muotoon
51                      int uuni = Convert.ToInt32(purkuaika.OVEN);
52
53                      purkuaika.OVEN = uuni;
54
55                      PurkuaikaData.Add(purkuaika);
56
57                  }
58                  //Asetetaan DataGrid -komponentteihin
59                  PurkuAjat.ItemsSource = PurkuaikaData;
60              }
```

KUVA 22. Purkuaikojen hakeminen ja syöttäminen DataGrid-komponenttiin

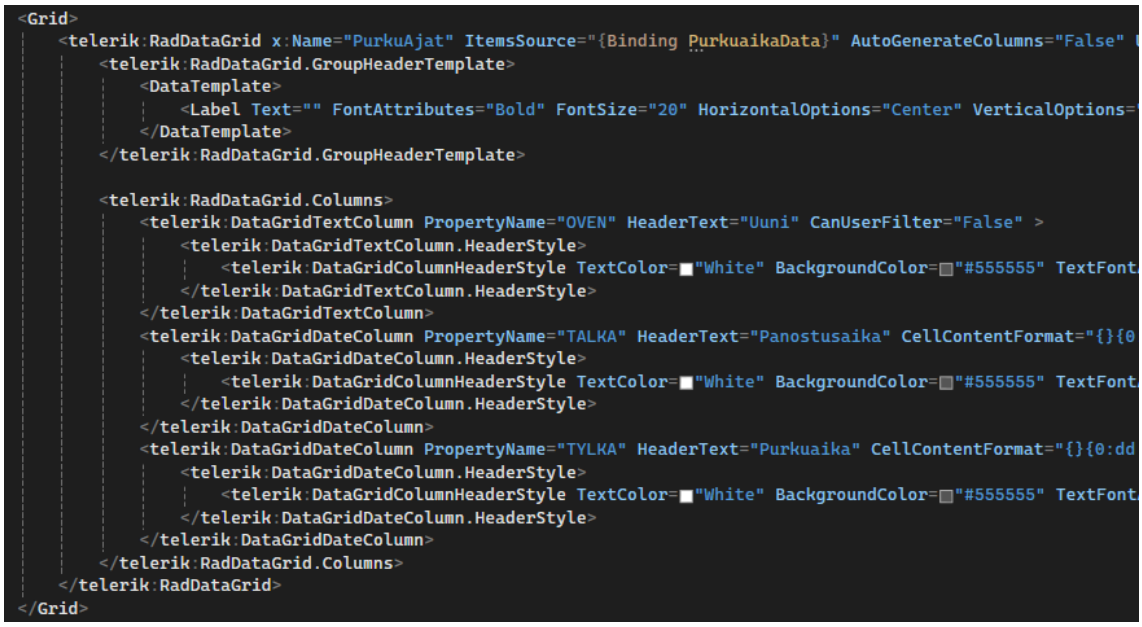
3.4 Näkymät

MainPage eli etusivulle tuli näkyväksi säätiedot, jotka tulevat tehtaan omalta sääasemalta. Säätiedot sisälsivät päivämäärän, lämpötilan, ilmankosteuden ja tuulennopeuden (kuva 23). Säätiedot tulostetaan allekkain tekstietiketteihin. Säätiedot haettiin eri tietokannasta kuin purkuajat, purkujen erot optimiin ja maksimitehot.



KUVA 23. Etusivun säätiedot

PurkuaikaView.xaml-tiedostoon loin Telerik-kirjaston DataGrid-taulukon. Uunien numeroiden lisäksi taulukon sarakkeisiin tuli näkymään uunien panostus- ja purkuajat. Sivulla pystyy selaamaan alaspäin kaikkia 70 uunia ja niiden tietoja. Sarakkeiden otsikot on värjätty harmaaksi, jotta ne erottuvat hyvin joukosta. (Kuva 24.)



KUVA 24. DataGrid-taulukko Purkuaika-sivulla

Käyttämällä ItemsSource-ominaisuutta voidaan määritellä lista kohteista, jotka näytetään DataGrid-taulukossa. Binding mahdollistaa taas tietojen sitomisen käyttöliittymäkomponenttiin sekä yhdistää sovelluksen datamallin ja käyttöliittymän mallit niin, että muutokset päivittyvät automaattisesti. Näin saadaan DataGrid-taulukko päivittymään tietokannan mukaan reaaliajassa. Taulukko on järjestetty purkuajan mukaan eli, panostusaika sarake kertoo uunin numeron mukaan, milloin uuni on panostettu ja purkuaika kertoo tulevaisuuteen, että mihin päivämäärään ja kellonaikaan uuni tulisi purkaa (kuva 25).

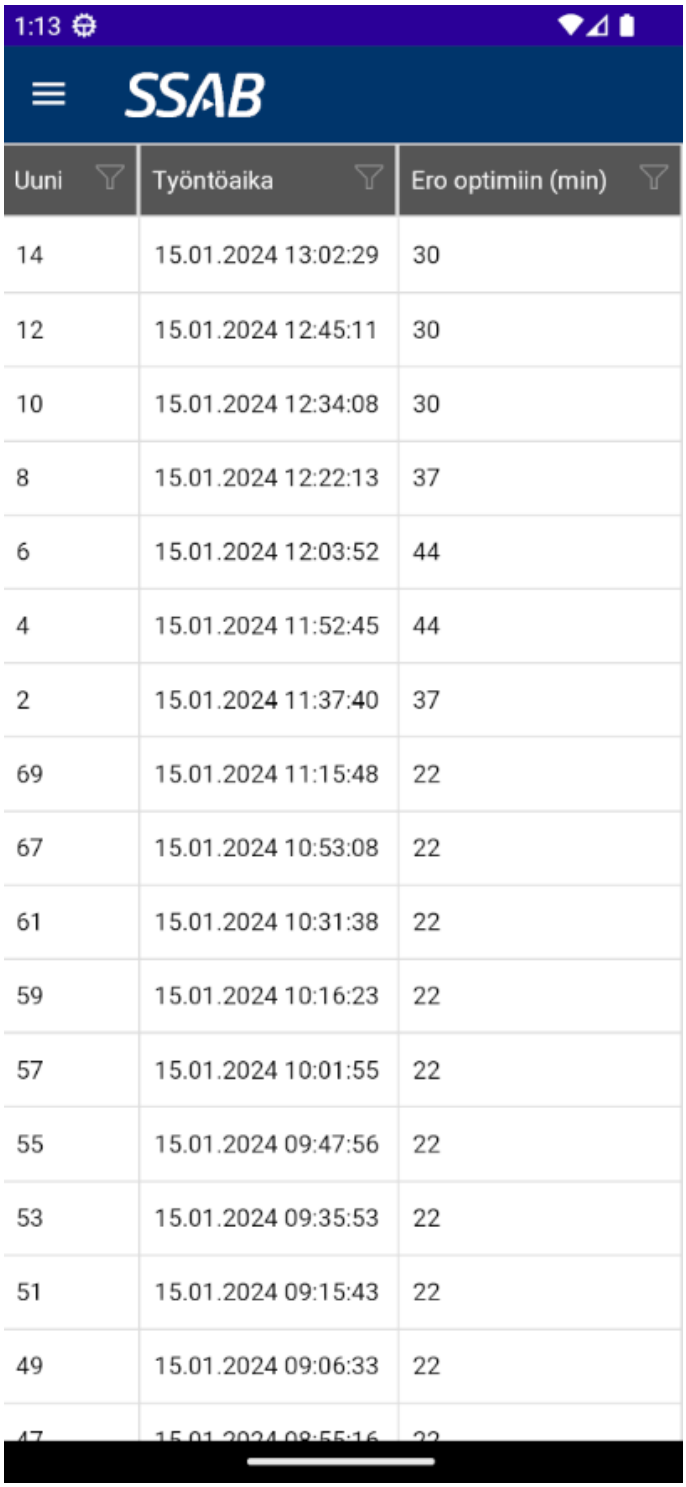
KUVA 25. Purkuaikataulu

Vuoronpurut ero optimiin sivulle tuli näkyväksi aamu- ja iltavuorojen purkamien uunien erot optimi purkuajasta. Nämä sijoitettiin kahteen erilliseen, mutta identtiseen DataGrid-taulukkoon, koska tiedot näihin tulevat eri kyselyillä. Taulukoihin rakennettiin sarakkeet uunien numeroille, uunien työntöajoille ja erot optimi purkuaikoihin minuutteina (kuva 26).

```
<Grid>
    <telerik:RadDataGrid x:Name="tyontoaikaAamu" ItemsSource="{Binding TyontoaikaAamu}" AutoGenerateColumns="False" UserGroupMode="
        <telerik:RadDataGrid.Columns>
            <telerik:DataGridTextColumn PropertyName="puuni" HeaderText="Uuni" CanUserFilter="False" >
                <telerik:DataGridTextColumn.HeaderStyle>
                    <telerik:DataGridColumnHeaderStyle TextColor="White" BackgroundColor="#555555" TextFontAttributes="Bold" />
                </telerik:DataGridTextColumn.HeaderStyle>
            </telerik:DataGridTextColumn>

            <telerik:DataGridDateColumn PropertyName="tylka" HeaderText="Työntöaika" CellContentFormat="{}{0:dd.MM.yyyy HH:mm:ss}"
                <telerik:DataGridDateColumn.HeaderStyle>
                    <telerik:DataGridColumnHeaderStyle TextColor="White" BackgroundColor="#555555" TextFontAttributes="Bold"/>
                </telerik:DataGridDateColumn.HeaderStyle>
            </telerik:DataGridDateColumn>

            <telerik:DataGridNumericalColumn PropertyName="ero_optimiin" HeaderText="Ero optimiin (min)" CanUserFilter="False" >
                <telerik:DataGridNumericalColumn.HeaderStyle>
                    <telerik:DataGridColumnHeaderStyle TextColor="White" BackgroundColor="#555555" TextFontAttributes="Bold"/>
                </telerik:DataGridNumericalColumn.HeaderStyle>
                <telerik:DataGridNumericalColumn.CellContentTemplate>
                    <DataTemplate>
                        <Grid BackgroundColor="{Binding ero_optimiin, Converter={StaticResource EroOptimiinConverter}}">
                            <Label Text="{Binding ero_optimiin}"
                                   TextColor="Black"
                                   VerticalOptions="Center"
                                   HorizontalOptions="Start"
                                   FontSize="15"
                                   Padding="7,0,0,0"/>
                        </Grid>
                    </DataTemplate>
                </telerik:DataGridNumericalColumn.CellContentTemplate>
            </telerik:DataGridNumericalColumn>
        </telerik:RadDataGrid.Columns>
    </telerik:RadDataGrid>
```

KUVA 26. Vuoronpurut ero optimiin sivun aamuvuoron DataGrid-elementti

Taulukot on tehty samalla tavalla kuin Purkuaikataulu eli käyttämällä ItemsSource- ja Binding-ominaisuuksia. Taulukko tyhjentyy aina kuudelta aamulla sekä illalla, ja sen jälkeen niihin alkaa tulemaan uunin numeron mukaan tietoja siinä järjestyksessä, mikä on seuraavaksi työntövuorossa oleva uuni. Kuvassa 27 näkyy esimerkki, jossa erot optimiin sarakkeessa kaikki työnnöt ovat reilusti myöhässä optimaalisesta aikataulusta.



KUVA 27. Vuoronpurut ero optimiin näkymä

Maksimiteho näkymässä on tehty pylväskaavio työntö- ja taaksetehojen maksimitehoille. Sivun ylälaitaan on luotu kaksi ListPicker-elementtiä, joista ensimmäisestä voi valita patterin 2 uunit 1-35 ja toisesta patterin 1 uunit 36-70 (kuva 28). Pylväskaavio päivittyy uunin numeron valinnan mukaan.

```
<ScrollView>
    <StackLayout>
        <Grid>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="*" />
                <ColumnDefinition Width="*" />
            </Grid.ColumnDefinitions>
            <telerik:RadListPicker x:Name="listPickerPatteri2" Padding="15"
                                   ItemsSource="{Binding patteri2Uunit}"
                                   DisplayMemberPath="puuni"
                                   SelectionChanged="ListPickerUuni_SelectionChanged"
                                   PlaceholderTemplate="{StaticResource placeholderTemplate2}" >
                <telerik:RadListPicker.PopupSettings>
                    <telerik:PickerPopupSettings HeaderTemplate="{StaticResource headerTemplate}"
                                                 FooterTemplate="{StaticResource footerTemplate}" />
                </telerik:RadListPicker.PopupSettings>
            </telerik:RadListPicker>

            <telerik:RadListPicker x:Name="listPickerPatteri1" Padding="15"
                                   ItemsSource="{Binding patteri1Uunit}"
                                   DisplayMemberPath="puuni"
                                   Grid.Column="1"
                                   SelectionChanged="ListPickerUuni2_SelectionChanged"
                                   PlaceholderTemplate="{StaticResource placeholderTemplate1}" >
                <telerik:RadListPicker.PopupSettings>
                    <telerik:PickerPopupSettings HeaderTemplate="{StaticResource headerTemplate}"
                            FooterTemplate="{StaticResource footerTemplate}" />
                </telerik:RadListPicker.PopupSettings>
            </telerik:RadListPicker>

        </Grid>
```

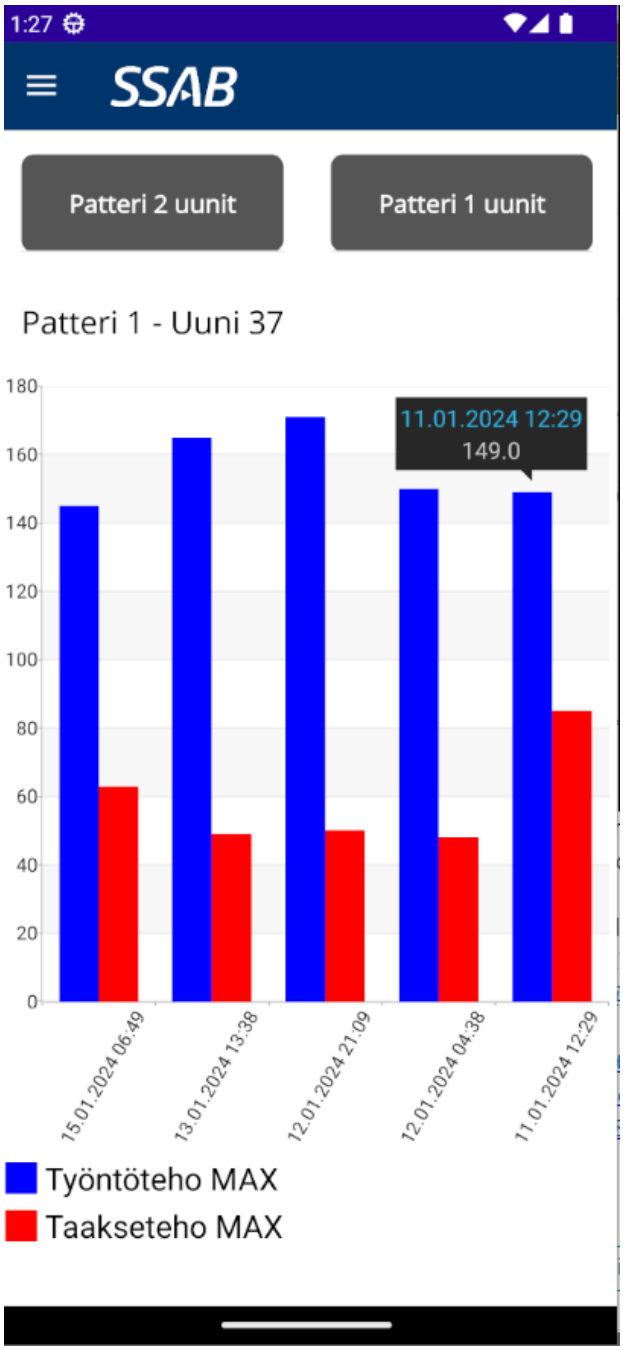
KUVA 28. ListPicker-elementit uuneille

RadCartesianChart on Telerik-kirjaston pylväskaavio. Tässä kaaviossa käytettiin myös aikaisemmin mainitsemalla tavalla ItemsSource- ja Binding-ominaisuuksia. ChartToolTipBehavior aktivoituu käyttäjän painaessa pylvästä, jolloin näkyviin tulee sen pylvään tiedot. Palette-ominaisuudella määritellään, minkä värisiä pylväät ovat. Kuvassa 29 on kokonaisuudessaan pylväskaavion ja sen ulkoasun määrittäminen.

```
        <telerik:RadCartesianChart x:Name="barchartMaksimit" HeightRequest="500" >
            <telerik:RadCartesianChart.HorizontalAxis>
                <telerik:CategoricalAxis LabelFitMode="Rotate" />
            </telerik:RadCartesianChart.HorizontalAxis>
            <telerik:RadCartesianChart.VerticalAxis>
                <telerik:NumericalAxis Minimum="0" x:Name="verticalAxis" />
            </telerik:RadCartesianChart.VerticalAxis>
            <telerik:RadCartesianChart.Series>
                <telerik:BarSeries ItemsSource="{Binding maksimiTehot}" CombineMode="Cluster" DisplayName="Työntöteho MAX" >
                    <telerik:BarSeries.ValueBinding>
                        <telerik:PropertyNameDataPointBinding PropertyName="ttmax" />
                    </telerik:BarSeries.ValueBinding>
                    <telerik:BarSeries.CategoryBinding>
                        <telerik:PropertyNameDataPointBinding PropertyName="tylkaFormatted" />
                    </telerik:BarSeries.CategoryBinding>
                </telerik:BarSeries>

                <telerik:BarSeries ItemsSource="{Binding maksimiTehot}" CombineMode="Cluster" DisplayName="Taakseteho MAX" >
                    <telerik:BarSeries.ValueBinding>
                        <telerik:PropertyNameDataPointBinding PropertyName="tatemax" />
                    </telerik:BarSeries.ValueBinding>
                    <telerik:BarSeries.CategoryBinding>
                        <telerik:PropertyNameDataPointBinding PropertyName="tylkaFormatted" />
                    </telerik:BarSeries.CategoryBinding>
                </telerik:BarSeries>
            </telerik:RadCartesianChart.Series>
            <telerik:RadCartesianChart.Grid>
                <telerik:CartesianChartGrid StripLinesVisibility="Y"
                                            MajorLinesVisibility="Y"
                                            MajorLineColor="DarkGray"
                                            MajorLineThickness="5" />
            </telerik:RadCartesianChart.Grid>

            <telerik:RadCartesianChart.ChartBehaviors>
                <!--    <telerik:ChartSelectionBehavior x:Name="selectionBehavior" DataPointSelectionMode="Single" -->
                <telerik:ChartTooltipBehavior TriggerMode="Tap" />
            </telerik:RadCartesianChart.ChartBehaviors>

            <telerik:RadCartesianChart.Palette>
                <telerik:ChartPalette>
                    <telerik:ChartPalette.Entries>
                        <telerik:PaletteEntry FillColor="Blue" StrokeColor="Blue" />
                        <telerik:PaletteEntry FillColor="Red" StrokeColor="Red" />
                    </telerik:ChartPalette.Entries>
                </telerik:ChartPalette>
            </telerik:RadCartesianChart.Palette>
        </telerik:RadCartesianChart>

        <telerik:RadLegend HeightRequest="200"
                           LegendItemFontColor="Black"
                           LegendItemFontSize="20"
                           LegendProvider="{x:Reference Name=barchartMaksimit}" />

    </StackLayout>
</ScrollView>
```

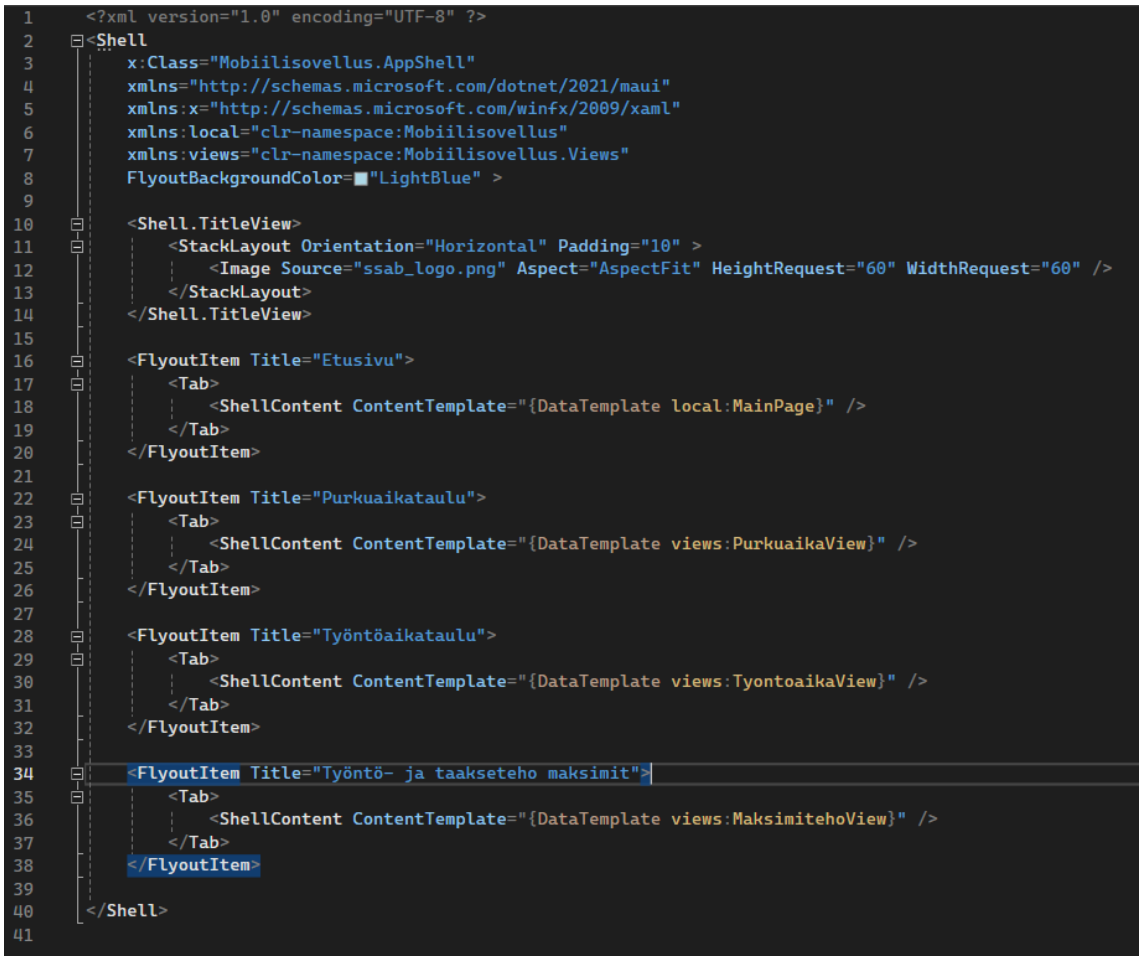
KUVA 29. Pylväskaavion määrittely

Kun uunin numero on valittu, pylväskaavioon päivittyy valitun uunin työntö- ja taakseteho maksimit sekä pylväiden alle työntöajat. Työntötehot on merkattu näkymään sinisellä ja taaksetehot punaisella. Pylvästä klikkaamalla voidaan nähdä pylvään tarkka arvo sekä sen työntöaika. (Kuva 29.)



KUVA 29. Työntö- ja taakseteho maksimeiden pylväskaavio

AppShell.xaml-tiedostossa on luotu navigointi mahdollisuus, joka on sijoitettuna sivujen yläkulmaan. Shell-elementti on MAUI-sovellusten peruskomponentti, joka tarjoaa yhteisen pohjan sovelluksen navigointirakenteelle. Tässä XML-tiedostossa määritellään myös sovelluksen perusarkkitehtuuri ja sen ulkoasu.



```
<?xml version="1.0" encoding="UTF-8" ?>
<Shell
    x:Class="Mobiilisovellus.AppShell"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:local="clr-namespace:Mobiilisovellus"
    xmlns:views="clr-namespace:Mobiilisovellus.Views"
    FlyoutBackgroundColor="LightBlue" >

    <Shell.TitleView>
        <StackLayout Orientation="Horizontal" Padding="10" >
            <Image Source="ssab_logo.png" Aspect="AspectFit" HeightRequest="60" WidthRequest="60" />
        </StackLayout>
    </Shell.TitleView>

    <FlyoutItem Title="Etusivu">
        <Tab>
            <ShellContent ContentTemplate="{DataTemplate local:MainPage}" />
        </Tab>
    </FlyoutItem>

    <FlyoutItem Title="Purkuaikataulu">
        <Tab>
            <ShellContent ContentTemplate="{DataTemplate views:PurkuaikaView}" />
        </Tab>
    </FlyoutItem>

    <FlyoutItem Title="Työntöaikataulu">
        <Tab>
            <ShellContent ContentTemplate="{DataTemplate views:TyontoaikaView}" />
        </Tab>
    </FlyoutItem>

    <FlyoutItem Title="Työntö- ja taakseteho maksimit">
        <Tab>
            <ShellContent ContentTemplate="{DataTemplate views:MaksimitehoView}" />
        </Tab>
    </FlyoutItem>

</Shell>
```

KUVA 30. AppShell.xaml-tiedosto perusarkkitehtuuria varten

4 POHDINTA

Opinnäytetyön tavoitteena oli rakentaa sovellus reaaliaikaiseen prosessin seurantaan niin Android- kuin iOS-laitteille. Sovelluksen alustana toimi Visual Studio Enterprise ja toteutukseen käytettiin .NET MAUI- ja ASP.NET Core Web API-viitekehyksiä. .NET MAUI-projekti oli tarkoitettu sovelluksen rakentamiseen, kun taas Web API -projekti toimi rajapintana palvelimelle, joka mahdollisti tietokannan tietojen välittämisen sovellukseen.

Kohtasin paljon haasteita opinnäytetyön tekemisessä johtuen siitä, että en ollut koskaan aikaisemmin käyttänyt .NET MAUIa, enkä myöskään ollut koskaan tehnyt web-rajapintaa tietojen käsittelemiseen. Haasteiden lisäksi selvisi myös, että sovelluksen asentaminen iOS-puhelimille vaatisi erilliset lisenssit Applelle sekä MacOS-laitteen. Tässä vaiheessa päätimme, että keskittyisimme saamaan toimivan sovelluksen ensisijaisesti Androidille. Jos ylimääräistä aikaa olisi jäänyt, olisimme vasta sitten alkanut suunnittelemaan erillisten lisenssien ja laitteiden hankkimista. Monialustaisena viitekehyksenä .NET MAUI ei ollutkaan ihan niin hyvä, kuin se alkuun antoi ymmärtää. iOS-kehityksen vaatimukset voisivat tulla esille selkeämmin Microsoftin sivuilla .NET MAUIn dokumentaatioissa.

Haasteista huolimatta sovelluksesta saatiin toimiva kokonaisuus Androidille, johon tuli tuotannon toiveiden mukaisesti säätiedot etusivulle, pylväskaavio työntö- ja taakseteho maksimeille ja taulukot purkuaikatiedoille sekä vuorojen purkuaikatietojen erot optimiin sivuille. Taulukoihin ja pylväskaavioon käytin Telerik-kirjaston komponentteja, joka helpotti työtä huomattavasti. Sovellus on rakennettu helppokäyttöiseksi, jotta se palvelee tarkoitustaan reaaliaikaisena seurantatyökaluna mahdollisimman hyvin. Taulukot ovat yksinkertaisia ja selkeitä sekä navigointi sivujen välillä on nopeaa.

Opinnäytetyön prosessin tärkeimpinä oppeina pidän haasteiden selvittämistä sekä uusien tekniikoiden opettelemista. Microsoftin teknologiat ovat helppokäyttöisiä ja ne toimivat hyvin, joten niiden opetteleminen ja käyttäminen oli erittäin antoisaa. Varsinkin Microsoftin Visual Studio toimii tehokkaana työkaluna esimerkiksi sovelluskehityksessä. Opin myös, että isoissa projekteissa kuten opinnäytetyössä, kommunikaatio ja projektinhallinta ovat tärkeä osa prosessia.

Mobiilisovellusten kehittämisessä on omat haasteensa, mutta nopeakäyttöisyyden vuoksi ne voivat olla erittäin hyödyllisiä tällaisessa tuotantoympäristössä. Ympärivuorokautiseen reaaliaikaiseen prosessin seurantaan ei ole oikein muita vaihtoehtoja kuin mobiilisovellus, koska nykyaikana puhelimet ovat aina mukana. Sen avulla voidaan parhaimmillaan estää tuotantokatkokset, koska tuotannossa tapahtuvat poikkeamat voidaan havaita nopeasti sovelluksen avulla.

Jatkokehityksenä mobiilisovellukseen tekisin lisää sivuja tuotannolle tärkeistä mittauksista ja tiedoista, kuten esimerkiksi uunien kuvat ja tiedot taulukkona. Sovelluksen voisi myös laajentaa iOS-laitteille, jotta sen käyttäminen ei olisi riippuvainen puhelimen käyttöjärjestelmästä. Sovelluksen pohjaa voisi myös hyödyntää tehtaan muilla osastoilla, koska sen sisältö on helppo muuttaa myös muille osastoille soveltuvaksi.

LÄHTEET

Churchville, Fred 2021. User interface (UI). TechTarget. Hakupäivä 9.10.2023. https://www.techtarget.com/searchapparchitecture/definition/user-interface-UI#:~:text=The%20user%20interface%20(UI)%20is,an%20application%20or%20a%20website..

Danyil, Martin 2022. How to build a multi-platform app with .NET MAUI. Fabrity. Hakupäivä 7.11.2023. https://fabrity.com/blog/how-to-build-a-multi-platform-app-with-net-maui/.

Halonen, Piret 2013. Ulkonäytön visualisointi ja tiedonsiirto ohjelmoitavista logiikoista. Oulun seudun ammattikorkeakoulu. Tietotekniikan koulutusohjelma. Opinnäytetyö. Hakupäivä 18.9.2023. https://www.theseus.fi/bitstream/handle/10024/65412/halonen_piret.pdf?sequence=1&isAllowed=y.

Hurja 2021. Käyttöliittymäsuunnittelu vaatii teknistä ja visuaalista osaamista. Hakupäivä 9.10.2023. https://www.hurja.fi/blogi/kayttoliittymasuunnittelu-vaatii-teknista-ja-visuaalista-osaamista/#:~:text=K%C3%A4ytt%C3%B6liittym%C3%A4%20(user%20interface%2C%20UI),UX%2Dsuunnittelu%20tarkoittaa%20k%C3%A4ytt%C3%B6kokemuksen%20suunnittelua..

Mannermaa, Tiia 2012. Koksaamon viikkopalaverin tuotantoraportit. Oulun seudun ammattikorkeakoulu. Tietotekniikan koulutusohjelma. Opinnäytetyö. Hakupäivä 18.9.2023. https://www.theseus.fi/bitstream/handle/10024/40640/Mannermaa_Tiia.pdf?sequence=1&isAllowed=y.

Microsoft 2022a. Kuvakaappaus. Model-View-ViewModel (MVVM). Hakupäivä 14.11.2023. https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm.

Microsoft 2022b. Kuvakaappaus. XAML code editor. Hakupäivä 6.12.2023. https://learn.microsoft.com/en-us/visualstudio/xaml-tools/xaml-code-editor?view=vs-2022.

Microsoft 2023a. What is .NET MAUI?. Hakupäivä 21.9.2023. https://learn.microsoft.com/en-us/dotnet/maui/what-is-maui.

Microsoft 2023b. Get started with XAML. Hakupäivä 20.9.2023. https://learn.microsoft.com/en-us/dotnet/maui/xaml/fundamentals/get-started.

Microsoft 2023c. Build. Test. Deploy. Hakupäivä 4.10.2023. https://dotnet.microsoft.com/en-us/.

Microsoft 2023d. App lifecycle. Hakupäivä 7.11.2023. https://learn.microsoft.com/en-us/dotnet/maui/fundamentals/app-lifecycle.

Microsoft 2023e. Controls by Category. Hakupäivä 7.11.2023. https://learn.microsoft.com/en-us/dotnet/desktop/wpf/controls/controls-by-category?view=netframeworkdesktop-4.8.

Microsoft 2023f. Data Binding and MVVM. Hakupäivä 14.11.2023. https://learn.microsoft.com/en-us/dotnet/maui/xaml/fundamentals/mvvm.

Microsoft 2023g. Kuvakaappaus. Controls. Hakupäivä 14.11.2023. https://learn.microsoft.com/en-us/dotnet/maui/user-interface/controls/?view=net-maui-8.0.

Mono 2023. About Mono. Hakupäivä 9.10.2023. https://www.mono-project.com/docs/about-mono/.

Pluszczewska, Bianka 2023. .NET MAUI in a Nutshell – Everything You Need to Know in 2023. Brainhub. Hakupäivä 9.10.2023. https://brainhub.eu/library/net-maui-in-nutshell.

Sharma, Anoop Kumar 2023. Project Structure of .NET MAUI Application. C#Corner. Hakupäivä 7.11.2023. https://www.c-sharpcorner.com/article/project-structure-of-net-maui-application/.

SSAB 2023. SSAB:n Raahen tehdas. Hakupäivä 18.9.2023. https://www.ssab.com/fi-fi/ssab-konserni/tietoja-ssabsta/tuotantopaikkakunnat-suomessa/raahe.

Telerik 2023a. Welcome to Telerik UI for .NET MAUI. Hakupäivä 26.10.2023. https://docs.telerik.com/devtools/maui/introduction.

Telerik 2023b. The Most Comprehensive UI Library of .NET MAUI Controls. Hakupäivä 26.10.2023. https://www.telerik.com/maui-ui.

Telerik 2023c. .NET MAUI ListView. Hakupäivä 26.10.2023. https://www.telerik.com/maui-ui/listview.

Telerik 2023d. .NET MAUI Bar Chart. Hakupäivä 26.10.2023. https://www.telerik.com/maui-ui/bar-chart.

Telerik 2023e. .NET MAUI Chart Numerical Axis. Hakupäivä 26.10.2023. https://docs.telerik.com/devtools/maui/controls/chart/axes/numerical-axis?_ga=2.12312572.995779848.1698305849-1507686913.1698305849&_gl=1*hch529*_gcl_au*OTc2NTMwOTQ5LjE2OT-gzMDU4NDk.*_ga*MTUwNzY4NjkxMy4xNjk4MzA1ODQ5*_ga_9JSNBCSF54*MTY5OD-MwNTg0OC4xLjEuMTY5ODMxMTYwNC42MC4wLjA.#net-maui-chart-numerical-axis.

TechTarget 2019. Model-View-ViewModel (MVVM). Hakupäivä 14.11.2023. https://www.techtarget.com/whatis/definition/Model-View-ViewModel.

Tizen 2023. Define UI Components in XAML. Hakupäivä 25.10.2023. https://docs.tizen.org/application/dotnet/guides/user-interface/nui/xaml/ui-component-in-xaml/.

Wikipedia 2023. SSAB. Hakupäivä 18.9.2023. https://fi.wikipedia.org/wiki/SSAB.