Lauri Nenonen

Summatiivisen harjoituksen kehittäminen Qt Basics -verkko-opintojaksoon

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

18.6.2025

Tiivistelmä

Tekijä:	Lauri Nenonen
Otsikko:	Summatiivisen harjoituksen kehittäminen Qt Basics -verkko-opintojaksoon
Sivumäärä:	22 sivua + 0 liitettä
Aika:	18.6.2025
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Älykkäät IoT-järjestelmät
Ohjaajat:	Osaamisaluejohtaja Janne Salonen

Tämä insinöörityö käsittelee summatiivisen ohjelmointiharjoituksen kehittämistä Metropolian Qt-ohjelmoinnin verkko-opintojaksolle. Työssä vertaillaan Qt-sovelluskehystä kahteen merkittävään vaihtoehtoon, Electroniin ja GTK:hon, keskittyen erityisesti käyttöliittymien suorituskykyyn, ulkoasuun ja kehitystyökaluihin. Qt:n vahvuuksia havainnollistetaan käytännönläheisen ”To-Do”-sovelluksen kautta, jolla perehtytään QML-kielen ominaisuuksiin, kuten deklaratiiviseen ohjelmointiin, signaaleihin ja animaatioihin. Tulosten perusteella Qt soveltuu erinomaisesti suorituskykyä vaativiin monialustaisiin projekteihin, erityisesti sulautettuihin järjestelmiin ja teollisuussovelluksiin.

Avainsanat: Qt, Electron, GTK, GUI, QML

Abstract

Author:	Lauri Nenonen
Title:	Development of a summative exercise for online Qt Basics programming course
Number of Pages:	22 pages + 0 appendices
Date:	18 June 2025
Degree:	Bachelor of Engineering
Degree Programme:	Information Technology
Professional Major:	Smart IoT Systems
Supervisors:	Osaamisaluejohtaja Janne Salonen

This thesis focuses on developing a summative programming exercise for an online Qt programming course for Metropolia. The study compares the Qt application framework with two significant alternatives, Electron and GTK, emphasizing aspects such as UI performance, appearance, and development tools. Qt's strengths are illustrated through a practical “To-Do” application, enabling students to explore essential QML features, including declarative programming, signals, and animations. The findings indicate that Qt is particularly suited for performance-critical cross-platform applications, notably in embedded systems and industrial software.

Keywords: Qt, Electron, GTK, GUI, QML

Sisällys

Lyhenteet

API: *Application Programming Interface*. Rajapinta, joka mahdollistaa ohjelmien kommunikoinnin keskenään.

GTK: *GIMP ToolKit*. avoimen lähdekoodin graafinen käyttöliittymäkirjasto. Se on kirjoitettu C-kielellä ja perustuu GObject-oliojärjestelmään, tarjoten joustavan tavan kehittää monialustaisia käyttöliittymiä.

GUI: *Graphical User Interface*. Visuaalinen käyttöliittymä, jonka kautta käyttäjä ohjaa ohjelmaa.

IDE: *Integrated Development Environment*: Ohjelmointiympäristö, joka yhdistää tyypillisesti koodieditorin, kääntäjän ja virheenkorjaimen yhdeksi kokonaisuudeksi.

MVC: *Model-View-Controller:* ohjelmistoarkkitehtuurimalli, jossa sovelluksen data (malli), käyttöliittymä (näkymä) ja käsittelylogiikka (ohjain) ovat eriytettyjä. Tämä helpottaa muun muassa käyttöliittymän kehittämistä erikseen datan käsittelystä.

QML: *Qt Meta-object Language:* Korkean tason deklaratiivinen kieli Qt Quick -käyttöliittymien määrittelyyn, jossa käyttöliittymän rakenne ja toiminta kuvataan visuaalisina komponentteina ja niiden suhteina.

Qt: *Qt*: monialustainen C++-pohjainen kehityskehys, jota käytetään graafisten käyttöliittymien sekä muiden sovellusten rakentamiseen.

Widgetti: *Widget*. Valmis graafinen käyttöliittymäelementti, kuten nappi, valikko tai liukusäädin.

1 Johdanto

2 Qt

Qt on monialustainen sovelluskehityskehys, jota käytetään erityisesti graafisten käyttöliittymien (GUI) kehittämiseen. Qt on laajalti käytetty teollisuudessa muun muassa sulautetuissa järjestelmissä, työpöytäsovelluksissa ja yhä enenevissä määrin myös mobiililaitteilla. Qt sai alkunsa norjalaisesta yhtiöstä nimeltä Trolltech, joka julkaisi ohjelmiston ensimmäisen version vuonna 1995. Nokia osti Trolltechin vuonna 2008 ja käytti Qt:ta muun muassa Symbian- ja MeeGo-alustoilla. Vuonna 2011 Nokia myi Qt:n kehityksen Digialle, joka jatkoi työtä itsenäisesti. Vuonna 2016 Digia eriytti Qt-liiketoiminnan omaksi yhtiökseen, The Qt Companyksi, joka on sittemmin kehittänyt Qt:ta eteenpäin ja listautunut Helsingin pörssiin.

Qt sai laajaa huomiota siitä, että se toimi pohjana suositussa KDE-työpöytäympäristössä Linuxissa vuodesta 1996 lähtien. Tuolloin Qt sai kritiikkiä suljetusta lähdekoodistaan, minkä seurauksena kilpaileva GTK-pohjainen GNOME-työpöytäympäristö sai alkunsa, jotta olennainen osa Linux-työpöytää ei olisi riippuvainen suljetun lähdekoodin ohjelmasta. Tämän paineen seurauksena lopulta myös Qt päätyi avoimeen lähdekoodiin. [1]

2.1 Arkkitehtuuri ja ominaisuudet

Qt on olio-ohjelmointia hyödyntävä ohjelmointi, joka perustuu C++:aan. Modulaarisena sillä on ydinosa Qt Core, joka sisältää olio-ohjelmoinnin peruspalikat kuten signaalit, säikeet ja tapahtumankäsittelyn ja sen päällä toimivat muut käyttöliittymämoduulit. Qt Widgets -moduuli sisältää perinteiset ikkuna- ja kontrollikomponentit sovelluksiin, kun taas Qt Quick -moduulin avulla voi rakentaa ohjelmia modernimmin deklaratiivisella QML-kielellä, joka muistuttaa JavaScriptiä. Widget-komponentteja ja QML-kuvauskielellä tehtyjä osioita voi myös yhdistää tarpeen mukaan. Esimerkiksi luokkaa QQuickWidget voi käyttää siltana,

jonka avulla QML-pohjaisen näkymän voi upottaa Qt Widget -pohjaiseen sovellukseen. [1][2]

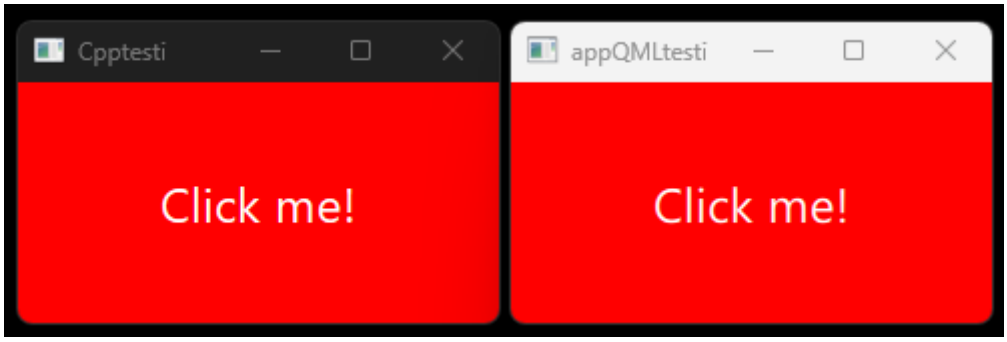
Esimerkkikoodi 1:ssä on toteutettu yksinkertainen QWidget-pohjainen ikkuna, jossa QLabel-komponentti näyttää keskellä tekstiä. Ikkuna on hiiren klikkauksille interaktiivinen ja vaihtaa väriä. C++-koodi antaa suuren hallinnan ohjelman rakenteeseen ja toimintaan, mutta vaatii enemmän koodia varsinkin ohjelman kasvaessa. Esimerkkikoodi 2 toteuttaa lähes saman ohjelman QML:llä. Koodia on vähemmän, joskin näin pienessä ohjelmassa ei kovin merkittävästi. Selkeitä etuja ovat kuitenkin koodin selkeys ja modulaarisuus, koska sovelluksen toimintalogiikka ja ulkoasu ovat eriytetty ja koodin rakenne on kuvaavampi. Jatkokehitys sujuisi näin ollen helpommin.

```
#include <QApplication>
#include <QWidget>
#include <QLabel>
#include <QMouseEvent>
#include <QVBoxLayout>

class ToggleWidget : public QWidget {
    QLabel *label;
    bool clicked = false;

public:
    ToggleWidget() {
        label = new QLabel("Click me!", this);
        label->setAlignment(Qt::AlignCenter);
        label->setStyleSheet("color: white; font-size: 24px;");

        auto layout = new QVBoxLayout(this);
        layout->addWidget(label);
        setLayout(layout);

        setStyleSheet("background-color: red;");
        resize(240, 120);
    }

protected:
    void mousePressEvent(QMouseEvent *) override {
        clicked = !clicked;
        label->setText(clicked ? "You clicked!" : "Click me!");
        setStyleSheet(clicked
            ? "background-color: lightblue; color: white; font-size:
24px;"
            : "background-color: red; color: white; font-size:
24px;");
    }
};

int main(int argc, char *argv[]) {
    QApplication app(argc, argv);
    ToggleWidget window;
    window.show();
    return app.exec();
}
```

Esimerkkikoodi 1. Interaktiivinen ikkuna toteutettuna C++:lla.

```
import QtQuick
import QtQuick.Controls
import QtQuick.Window

ApplicationWindow {
    visible: true
    width: 240
    height: 120

    Rectangle {
        id: background
        anchors.fill: parent
        color: background.clicked ? "lightblue" : "red"
        property bool clicked: false

        Text {
            anchors.centerIn: parent
            text: background.clicked ? "You clicked!" : "Click me!"
            font.pixelSize: 24
            color: "white"
        }

        MouseArea {
            anchors.fill: parent
            onClicked: background.clicked = !background.clicked
        }
    }
}
```

Esimerkkikoodi 2. Interaktiivinen ikkuna toteutettuna QML:lla.



Kuva 1. Yksinkertainen interaktiivinen ikkuna C++- ja QML-toteutuksella.

2.2 Suorituskyky

C++-pohjaisena kehyksenä Qt-sovellukset käännetään natiiviksi koodiksi kullakin alustalla hyödyntäen alla olevan käyttöjärjestelmän tarjoamia rajapintoja ilman tulkintakerroksia tai virtuaalikonetta, jolloin suorituskyky on hyvä ja

käyttäjän syötteillä on nopea vaste. Luotaessa Qt Widgets -sovelluksia voidaan kutsua käyttöjärjestelmän omia natiiveja komponentteja kuten esimerkiksi Windowsin nappeja, jolloin niiden ulkonäkö ja käyttäytyminen on hyvin saumatonta. Qt Widgets soveltuu erityisen hyvin monimutkaisiin ja raskaisiin työpöytäsovelluksiin tehokkaan resurssienkäyttönsä vuoksi. Qt Company panostaa kuitenkin enemmän Qt Quick:iin, joka ei toimi aivan yhtä integroidusti alla olevaan käyttöjärjestelmään, mutta mahdollistaa paremmin modernit ominaisuudet kuten skaalautuminen kosketuslaitteille ja sulavat animaatiot. [2][3]

2.3 Qt Creator

Qt-ekosysteemiin kuuluu joukko työkaluja, jotka tukevat sovellusten kehitystä ja nopeuttavat käyttöliittymien rakentamista. Näistä keskeisin on Qt Creator, joka on Qt:n oma monialustainen kehitysympäristö, joka tukee sekä C++- että QML-kehitystä. Qt Creatorissa on koodieditorin ja virheenkorjaimen ohella graafinen käyttöliittymäeditori: Qt Designer. Sillä voi rakentaa käyttöliittymiä vedä-ja-pudota-menetelmällä, muokata layoutit, luoda signaaliyhteydet ja nähdä muutokset reaaliajassa QML live preview -ominaisuudella aina kun QML-tiedostoa muokataan ja tallennetaan. Tämä nopeuttaa käyttöliittymän iterointia: kehittäjä näkee välittömästi miltä muutos kuten uusi komponentti, muuttunut väri tai layout, näyttää sovelluksessa ilman täyttä sovelluksen kääntämistä uudelleen. Lisäksi Qt Creator tarjoaa QML-debuggerin ja -profilerin, joilla voi analysoida virheitä ja suorituskykyä QML-koodissa. [4]

Muihin Qt-työkaluihin kuuluu muun muassa Qt Linguist, joka auttaa sovellusten tekstien kääntämisessä lokalisaatiota varten, Qt Assistant dokumentaation selaamiseen sekä komentorivityökalu qmake. Historiallisesti Qt on käyttänyt omaa qmake-työkaluaan eri alustoilla, mutta Qt 6 -sarjassa virallinen suositus on nykyään CMake. Qt Creator tukee molempia, mahdollistaen projektin kääntämisen helposti Windowsissa, Linuxissa tai macOS:ssä. Qt:n kehitysympäristö on siis varsin kattava: Qt Creator yhdistää työkalut suunnitteluun, koodaamiseen, kääntämiseen ja debuggaamiseen, ja mukana on erikoistyökalut käyttöliittymien visuaaliseen rakentamiseen, käyttöliittymäkoodin live-päivitykseen sekä

sovellusten lokalisointiin ja dokumentointiin. Nämä työkalut auttavat varmistamaan, että Qt-sovelluskehitys on sujuvaa alustasta riippumatta. [4][5]

2.4 Qt:tä käyttävät käyttöliittymät

Qt-teknologiaa käyttävät monet merkittävät ohjelmistot sekä avoimen lähdekoodin yhteisössä että kaupallisissa sovelluksissa. Suuri osa Qt:n perinteisistä työpöytäsovelluksista hyödyntää Qt Widgets -kirjastoa. Esimerkiksi avoimen lähdekoodin grafiikkaohjelmisto Krita käyttää Qt Widgets -kirjastoa käyttöliittymänsä toteuttamiseen, mahdollistaen natiivin näköisen ja tuntuisen käyttöliittymän eri käyttöjärjestelmillä. Vastaavasti Autodesk Maya -3D-ohjelmisto, joka on laajasti käytössä elokuva-, peli- ja animaatioteollisuudessa, perustuu Qt Widgetsiin.

Myös suosittu virtualisointiohjelma Oracle VirtualBox käyttää Qt Widgetsia graafisen käyttöliittymänsä rakentamiseen. KDE-työpöytäympäristössä Qt on täysin keskeisessä asemassa: lähes kaikki KDE-ohjelmat, kuten tiedostoselain Dolphin, musiikkisoitin Elisa ja tekstieditori Kate, perustuvat Qt Widgets -kirjastoon.

Qt Quick -teknologiaa käyttävät ohjelmistot ovat usein mobiili- ja sulautettujen järjestelmien sovelluksia, kuten monet autojen käyttöjärjestelmät. Erityisesti Mercedes-Benzin ja Teslan järjestelmät hyödyntävät Qt Quick -tekniikkaa sulavan animaation ja kosketuskäyttöliittymän rakentamisessa. [6][7]

3 Vaihtoehtoiset GUI-kehykset

3.1 Electron

Electron kehitettiin GitHubilla vuonna 2013 alun perin Atom-koodieditoria varten. Alusta julkaistiin avoimena lähdekoodina nimellä Atom Shell, ja se nimettiin myöhemmin Electroniksi. GitHubin omistuksen siirryttyä Microsoftille vuonna 2018 Electronin kehitys jatkui edelleen avoimena projektina. Nykyään Electronia ylläpitää OpenJS Foundationin alaisuudessa toimiva kehittäjäyhteisö, ja sitä

käytetään laajasti mm. Slackin, Discordin ja Visual Studio Coden kaltaisissa sovelluksissa. [8]

Electron on avoimen lähdekoodin kehitysalusta, joka mahdollistaa työpöytäsovellusten luomisen web-teknologioilla kuten HTML, CSS ja JavaScript. Electron yhdistää Chromium-selaimen renderöintimoottorin ja Node.js:n backend-teknologiat, mikä mahdollistaa täysimittaisten, alustariippumattomien sovellusten rakentamisen yhdellä koodipohjalla. Sen kehitti alun perin GitHub vuonna 2013, ja se on ollut suosittu erityisesti web-kehittäjien keskuudessa, jotka haluavat siirtää osaamisensa työpöytäsovelluksiin.

Electron-pohjaiset sovellukset ajetaan upotetulla Chromium-selaimella ja sen ominaisuudet toimivat Node.js:n avulla, mikä takaa yhtäläisen toimivuuden monenlaisilla alustoilla. Toisaalta koska jokaiseen Electron-sovellukseen pitää sisällyttää Chromium-selainmoottori ja Node.js-ajoympäristö, sovelluksen koko on suhteellisen iso kuten myös muut resurssivaatimukset.

Electronin vahvuuksia ovat kehityksen helppous web-kehittäjille, laaja yhteisö ja valmiit komponentit. Haittapuolena Electron-sovellukset ovat yleensä raskaampia, sillä jokainen sovellus sisältää oman selaimensa ja Node.js-ympäristönsä. [9][10]

3.2 GTK

GTK kehitettiin alun perin vuonna avoimeen lähdekoodiin perustuvaa kuvankäsittelyohjelmaa, 1997 GNU Image Manipulation Programia (GIMP) varten. Toolkitin kehitys eriytyi nopeasti laajempaan käyttöön, ja sitä alettiin käyttää useissa muissa vapaissa ohjelmissa. GTK:n kehityksestä vastaa pääasiassa GNOME-projekti. Toolkit on kehittynyt useiden versioiden kautta, ja nykyään se tukee useita käyttöjärjestelmiä ja käyttöliittymätyylejä, erityisesti Linuxissa.

GTK:n keskeinen ominaisuus on sen oliopohjainen arkkitehtuuri C-kielessä. Koska C-kieli ei natiivisti tue olio-ohjelmointia, GTK hyödyntää GLib-kirjaston

GObject-järjestelmää, joka tarjoaa olioiden perinnän, polymorfismin ja signaalimekanismin C-kielellä. Käytännössä jokainen käyttöliittymäelementti on GObject-olio. GTK sisältää kattavan joukon valmiita Widgettejä: muun muassa napit, teksti- ja numerokentät sekä valintaikkunat. GTK 3:een sisältyy noin 200 erilaista Widget-luokkaa. Uusimmassa GTK4:ssä on myös modernisoitu grafiikkamoottori: piirto hoidetaan pääosin GPU-kiihdytettynä käyttäen GDK-rajapintaa, joka abstraktoi alustan piirtoikkunoinnin alla olevan käyttöjärjestelmän suoritettavaksi ja hyödyntää Cairo-kirjastoa 2D-piirtämiseen. GTK4:ssä on lisäksi kehityksessä oma scene graph -tyylinen renderöintimalli, joka on samankaltainen kuin Qt Quickissä.

Monialustaisuus on periaatteessa osa GTK:ta: se toimii Linuxin lisäksi Windowsissa ja macOS:ssä. Perinteisesti kuitenkin GTK:n vahvin tuki ja käyttökohteet ovat Linux-järjestelmissä. Windows- ja Mac-versiot ovat olemassa, mutta usein GTK-sovellukset näyttävät ja tuntuvat parhaimmilta GNOME-ympäristössä. Näin ollen tyypillinen GTK-sovellus voi näyttää vieraalta Windows-ympäristössä verrattuna aidosti Windowsin omilla rajapinnoilla tehtyihin sovelluksiin. Tästä huolimatta useita tunnettuja sovelluksia on tehty GTK:lla myös Windowsille, esimerkiksi GIMP-kuvankäsittelyohjelma ja Inkscape-vektorigrafiikkaohjelma. [11]

4 GUI-kehyksien keskinäinen vertailu

4.1 Arkkitehtuuri ja ominaisuudet

Qt ja GTK ovat molemmat perinteisiä käyttöliittymäkirjastoja, kun taas Electron perustuu selainteknologiaan. Qt on C++-pohjainen monipuolinen kehys, joka sisältää laajoja moduulikirjastoja kuten Qt Core, Qt GUI, Qt Quick/QML, Qt Network ja Qt Multimedia ja tukee signaali/slot-viestintäjärjestelmää sekä oliopohjaista laajennettavuutta. GTK puolestaan on ohjelmistokirjasto, jossa on valmiita C-kielisiä widgettejä, joille Glib-kirjasto mahdollistaa olio-ohjelmoinnin käyttämistä niin että esimerkiksi GtkButton-nappikomponentti voi periä ominaisuuksia GtkWidget-luokalta. Qt:lla on vahva tuki monelle alustalle, piirtäen sisältönsä natiivisti Qt Widgetsin C++:n avulla tai QML-kielellä rakennettuna Qt Quickillä,

mikä mahdollistaa laajat dynaamiset ominaisuudet erinäisillä käyttöliittymäkomponenteilla. [2][11][12]

4.2 Suorituskyky

Suorituskyky: Qt ja GTK edustavat perinteisiä natiivikirjastoja, jotka käännetään suoraan konekielelle ja ne käyttävät käyttöjärjestelmän tarjoamia rajapintoja tehokkaasti. Tämä näkyy yleensä korkeana suorituskykynä: oikein toteutettu Qt- tai GTK-sovellus pystyy hyödyntämään laitteistoa optimaalisesti ja käyttäjän tuntuma käyttöliittymään on viiveetön. Qt-sovellukset ovat todellisia natiivisovelluksia, joilla on natiivien ohjelmien suorituskyky ja ominaisuudet. Esimerkiksi Qt Quick -tekniikka voi hyödyntää grafiikkapiiriä raskaassa piirrossa (scene graph -pohjainen GPU-renderöinti), mikä mahdollistaa sulavat animaatiot ja kompleksitkin käyttöliittymät säilyttäen korkean kuvataajuuden. GTK puolestaan on tunnettu keveydestään: C-kielisenä suorituskyky on lähempänä rautatasoa, ja GTK4:stä lähtien piirrossa on mahdollisuus OpenGL-kiihdytykselle. GTK käyttää piirtämiseen välikerrosta GDK (GIMP Drawing Kit) joka käyttää allaolevan käyttöjärjestelmän API:a. GDK4:ssa on käytössä uudempi kerros GSK (GTK Scene Kit), jolla piirto on GPU-kiihdytettyä hyödyntämällä muun muassa OpenGL tai Vulkan. Sekä Qt:ssä että GTK:ssa on mahdollista kirjoittaa sovelluksen kriittiset osat tarvittaessa alhaisen tason kielellä (C/C++), mikä mahdollistaa pullonkaulojen optimoinnin vaativissa sovelluksissa. [13]

Electronin suorituskyky on lähtökohtaisesti heikompi, koska sen alla pyörii kokonainen verkkoselaimen moottori. Jokainen käyttöliittymäelementti on HTML-DOM elementti, jonka piirtämisestä huolehtii Chromium. Vaikka modernit selaimet on optimoitu, on niiden ylimääräinen kerros aina natiivia raskaampi. Electron-sovellusten on raportoitu kuluttavan merkittävästi prosessoritehoa esimerkiksi taustalla, mikä voi johtua Chromen jatkuvasta toiminnasta muun muassa roskien keruussa. Muistinhallinnassa Qt ja GTK:lla tottelevat kehittäjää, kun taas Electronissa käytetään JavaScriptin roskankeruumekanismeja, jotka eivät aina vapauta muistia optimaalisesti. Yksinkertaisissa testeissä on havaittu, että esimerkiksi jo yksinkertainen “Hello World” -tyyppinen sovellus käyttää

merkittävästi enemmän muistia Electronilla. Voidaan puhua yli sadasta megatavusta siinä missä Qt ja GTK pärjäävät muutamalla megatavulla. [10]

Yhteenvetona suorituskyvystä voidaan todeta, että Qt on kolmikon tehokkain raskaassa käytössä kuten 3D-grafiikassa ja videonkäsittelyssä. GTK on myös erittäin kevyt ja nopea 2D-rajapintojen piirtämisessä mutta se ei ehkä skaalaudu aivan yhtä hyvin massiivisiin sovelluksiin (Qt:n laaja moduulikirjasto antaa etua), mutta ydin on nopea. Electron jää jälkeen suorituskyvyssä. Se sopii moniin käyttötapauksiin riittävällä tasolla, mutta sen kanssa on vaikea päästä samaan resurssitehokkuuteen. Jos lopputuotteessa keskeistä on pieni muistinkulutus, vähäinen prosessorikuorma ja sulava natiivikokemus, Qt tai GTK ovat parempia valintoja. [12][14]

4.3 Käyttöliittymän ulkoasu

Käyttöliittymissä on toisinaan haluttua saada se sopeutumaan alla olevan käyttöjärjestelmän näköiseksi. Qt ja GTK tarjoavat paremmat valmiudet natiivin näköisille ja tuntuisille käyttöliittymille. Qt Widgets -sovellus esimerkiksi hyödyntää oletustyylinä isäntäkäyttöjärjestelmän teemoja (Windowsin tai Macin ikkunatyylit, KDE tai Gnome-tyylit Linuxissa), jolloin lopputulos sulautuu hyvin kyseisen alustan ohjelmiin. GTK-sovellukset sopivat erityisen hyvin GNOME:een, mutta Windowsissa ja macOS:ssä ne voivat näyttää hieman epätyypillisiltä, ellei teemaa erikseen soviteta. Qt Quick -sovellusten osalta Qt Quick Controls tarjoaa teemamekanismin, jolla voidaan valita joko alustakohtainen tai Qt:n oma tyyli. Vaihtoehtoisesti QML:llä on mahdollista tehdä täysin mukautetun näköinen käyttöliittymä. Qt antaa siis kehittäjälle valinnanvapauden: joko natiivi look tai yhtenäinen tyyli, joka näyttää samalta alustasta riippumatta.

Electronilla tehty sovellus puolestaan piirtää HTML/CSS-käyttöliittymänsä Chromium-selaimen sisällä, joten se näkyy samanlaisena käyttöjärjestelmästä riippumatta. Electron tarjoaa tarvittaessa myös API:t esimerkiksi natiivien valikkorivien luontiin ja ilmoitusten näyttöön, jolloin sovellus voi käyttää niiden näyttämiseen käyttöjärjestelmän omia komponentteja. Silti varsinainen pääikkunan

sisältö on web-sivu, joka ei automaattisesti noudata isäntäjärjestelmän UI-ohjeistuksia. Kehittäjän on erikseen tyyliteltävä sovellus näyttämään halutulta – tämä voi olla etu tai haitta. Usein Electron-sovellukset tuovat mukanaan täysin oman käyttöliittymätyylinsä. Esimerkiksi Slack ja Discord eivät yritä olla Windowsin tai macOS:n näköisiä, vaan niillä on brändätty UI, joka on saman näköinen käyttöjärjestelmästä riippumatta. Tässä mielessä Electron-appien natiivisuus on heikointa: ne eivät käyttäydy tai tunnu aina samalta kuin perinteiset työpöytäsovellukset. Esimerkiksi näppäinyhdistelmien toiminta, tekstikenttien kontekstivalikot tms. ovat joskus erilaisia, koska ne ovat selainympäristön määrittämiä. Qt- ja GTK-sovellukset sen sijaan käyttävät suoraan OS:n input- ja käyttöliittymäkomponentteja, joten niiden toiminta noudattaa oletuksia, jolloin esimerkiksi hiiren oikean painikkeen valikko tulee alla olevalta käyttöjärjestelmältä. [2][11][14]

4.4 Kehitystyökalut

Qt on pääasiassa C++-pohjainen kehys. Sovellusten ydinkoodi ja liiketoimintalogiikka kirjoitetaan yleensä C++:lla käyttäen Qt:n laajennettua oliojärjestelmää. Qt kuitenkin tarjoaa myös oman QML JavaScript-kielen käyttöliittymäkerrokseen, kuten edellä käsiteltiin. Niinpä Qt-kehittäjä voi yhdistellä C+ +:aa ja QML:ää tarpeen mukaan. Lisäksi Qt:sta on olemassa muita kielisidoksia: suosituimpia vaihtoehtoja on Python, joka on virallisesti tuettu vaihtoehto. Pythonilla voidaan kirjoittaa Qt-sovelluksia Python-sidoksien avulla. Vähemmän tuettuja sidoksia on olemassa myös kielille Java, PHP, ja C#. Pääasiallinen kieli on kuitenkin C++/QML yhdistelmä. Tämä vaatii kehittäjältä C++-osaamista, mikä on korkeampi vaatimus kuin pelkän skriptikielen hallitseminen. Toisaalta C++:n teho ja Qt:n tarjoamat makrot mahdollistavat tehokkaan ohjelmoinnin, kun kieli on opittu.

Qt-kehitystyökaluista merkittävin on Qt Creator -ohjelmointiympäristö. Qt Creatorissa on syvä integraatio Qt:n ominaisuuksiin: projektipohjat, graafinen suunnittelu (Qt Designer), automaattinen koodin täydennys Qt:n luokille, sisäänrakennettu dokumentaatio ja debuggaus, QML-profilointi. Lisäksi Qt-projekteja voi

kehittää muillakin työkaluilla; esimerkiksi Visual Studio -integraatio on saatavilla. Myös komentorivityökaluja kuten qmake ja cmake voi käyttää. Kokonaisuutena Qt tarjoaa suurillekin projekteille sopivan kehitysympäristön. [4]

GTK:n pääohjelmointikieli on perinteisesti C. Sovellukset kirjoitetaan C-kielellä kutsuen GTK-kirjaston funktioita ja luomalla GObject-tyyppisiä olioita. Matalan tason ohjelmointi on melko vaativaa ja GTK:lle onkin kehitetty laaja valikoima kielisidoksia. Esimerkiksi PyGObject mahdollistaa GTK-ohjelmoinnin Pythonilla; vastaavasti gtkmm on C++-luokkakirjasto GTK:lle, JavaScript (GJS) käytetään GNOME Shell -laajennuksissa, ja Rust (gtk-rs) on suosittu moderni tapa kehittää GTK:lla. Monet GNOME-projektit on viime aikoina tehty esimerkiksi Rustilla GTK4:ää käyttäen, koska Rust tarjoaa muistiturvaa ja tehokkuutta. Kielivalinta vaikuttaa myös kehitystyökaluihin: C-kehityksessä käytetään usein C:n yleistyökaluja (gcc/clang, GDB debugger). GNOME Builder IDE tukee erityisesti C- ja Rust-projekteja hyvin. Pythonilla kehittäessä voi käyttää vaikkapa Visual Studio Codea tai PyCharmia, joihin on saatavilla liitännäiset GObjecteja varten. GTK:lla on oma käyttöliittymäsuunnittelutyökalu Glade, jossa käyttöliittymää voidaan rakentaa napsimalla widgettejä paikoilleen. Glade kuitenkin tukee vain staattista käyttöliittymäkuvausta, logiikka on edelleen koodattava käsin. Uusissa GTK4-sovelluksissa on myös vaihtoehtoja kuten libadwaita, joka tarjoaa valmiita korkeantason komponentteja, mutta kehittäjä yleensä kirjoittaa itse sovelluksen rakenteen koodissa. Kehitystyökalut ovat vähemmän integroidut kuin Qt:lla: esimerkiksi erillinen graafinen debugger tai suorituskykyprofiili ei ole suoraan osa GTK-projektia, vaan käytetään yleisiä työkaluja. Dokumentaatio GTK:lle on saatavilla GNOME Developer Centerissä, ja se on kattavaa, joskin Qt:n dokumentaation ylistettyyn laajuuteen se ei aivan yllä. [15]

Electronin ohjelmointikieli on käytännössä JavaScript. Käyttöliittymä kirjoitetaan web-tekniikoilla: HTML kuvaa rakenteen, CSS tyylit ja JavaScript hoitaa dynaamisen toiminnan. Taustaprosessissa voi myös käyttää Node.js:n tuella muita kieliä, esimerkiksi moduuleja voi kirjoittaa C/C++:lla jos tarvitsee suorituskykyä, mutta valtaosa koodista on JS:ää. Electronin kehitystyökalut ovat pitkälti samat kuin web-kehityksessä: koodieditoreina Visual Studio Code, Atom, WebStorm

ovat yleisiä. Debuggaus tapahtuu usein Chromium-selaimen DevTools-työkalujen kautta. Tämä on web-kehittäjille tehokasta, koska hallittuja taitoja voi soveltaa. Asennuspakettien luomiseen on automaattisia työkaluja. Esimerkiksi electron-packager-työkalulla projektista voidaan luoda Windows-, Mac- ja Linux-kansiot, joissa on kaikki tarvittava ohjelman ajamiseen. Tämän jälkeen voidaan käyttää electron-builder-työkalua luomaan asennusohjelmat. Nämä työkalut ovat komentorivipohjaisia ja usein integroitu npm-skriptien kautta projektiin. Kehitysympäristön osalta Electron hyötyy vahvasti web-kehityksen valtavasta määrästä valmiita ratkaisuja: kehittäjät voivat käyttää esimerkiksi Reactin kehityskirjastoja, Reduxin devtoolsia tilanhallintaan ja selaimen automaattista sivupäivitystä sovellusta tehdessä. Web-teknologioiden hallitsijoille Electronin kehitysnopeus onkin erinomainen, koska tarvittavat työkalut ovat hyvin hiottuja vuosien myötä. Sen sijaan perinteiset C++/C ympäristöt Qt- ja GTK-kehityksessä edellyttävät kääntämistä ja debuggausta perinteisillä menetelmillä, mikä on hitaampaa kuin web-kehityksen välitön tulos. QML-kehityksessä Qt tarjoaa live reload -toimintoa Qt Quick Designerissa, mutta se ei ole yhtä saumaton kuin web-selaimen vastaava. Electronilla kehittäjä voi usein ladata sovelluksen uudelleen sekunnin murto-osissa muutoksen jälkeen, mikä tehostaa interaktiivista kehitystä. [9][10][12]

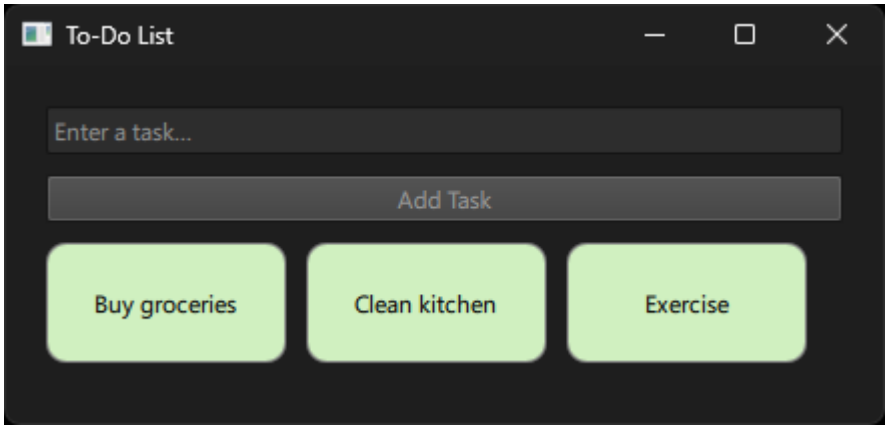
Yhteenvetona kielistä ja työkaluista: Qt vaatii QML:n hallitsemisen ja C++-osaaminen on eduksi, kun taas GTK vaatii C- tai muita sidoskieliä. Electron nojaa JavaScriptiin. Osaajia Electronin web/JS-teknologioihin on eniten tarjolla, kun taas C++/Qt-osaajia harvemmassa tämä voi vaikuttaa teknologiavalintaan organisaatiotasolla. Työkalut ovat kaikilla hyvät mutta erityyppiset: Qt:lla Qt Creator on hyvin kaiken tarvittavan keskittävä työkalu siinä missä GTK:lla kehitys on hajautetumpi useita erillisiä työkaluja hyödyntämällä. Electronilla nojaudutaan webmaailman työkaluihin kuten Chromiumin DevTools ja Node.js:n npm. Mikäli projekti arvostaa vahvaa IDE-tukea ja valmista suunnittelutyökalua, Qt on edukseen. Jos taas halutaan hyödyntää esimerkiksi olemassa olevaa web-projektin koodia, Electron on vahvoilla. GTK sijoittuu ehkä näiden väliin: se antaa monikielistä joustavuutta, mutta ei tarjoa yhtä paljon korkean tason kehitystyökaluja.

5 Summatiivinen QML-harjoitus

5.1 Harjoituksen kuvaus ja tavoitteet

Summatiivisen harjoituksen tavoitteena on vahvistaa keskeisiä Qt Quick -konsepteja, joita opiskelijat ovat oppineet Qt Basics -kurssilla. Näitä ovat deklaratiivinen käyttöliittymäsuunnittelu, dynaaminen tietojen sidonta, Qt Quick Controls -komponenttien käyttö sekä animaatioiden lisääminen. Opiskelijat luovat askel askeleelta yksinkertaisen mutta toimivan ”To-Do” -tehtävälistan QML-kielellä käyttäen Qt Creator -kehitysympäristöä. Harjoitus ei edellytä C++-ohjelmointia, joten se soveltuu hyvin myös aloittelijoille.

Kuvassa 2 näkyy tavoitteena oleva harjoitusohjelma, jossa tekstikenttään voi lisätä tehtäviä, jotka järjestyvät ikkunaan. Klikkaamalla tehtävää ne poistuvat ikkunasta määritellyllä animaatiolla.



Kuva 2. Harjoitusohjelma ”To-Do”-tehtävälista

5.2 Mitä harjoituksesta opitaan?

Harjoituksessa opitaan ymmärtämään ja hyödyntämään QML:n deklaratiivista lähestymistapaa, jossa käyttöliittymä määritellään selkeästi ja korkeatasoisesti ilman erillistä ohjelmallista komponenttien rakentelua. Opitaan myös käyttämään Qt:n signaaleja käyttöliittymän interaktiivisuuden ja reaktiivisuuden luomiseksi sekä hyödyntämään valmiita QtQuick.Controls -komponentteja

tehokkaan ja yhtenäisen käyttöliittymän rakentamiseen. Lisäksi harjoitus tutustuttaa QML:n animaatio-ominaisuuksiin, jotka lisäävät visuaalista sulavuutta ja parantavat käyttökokemusta ilman monimutkaista ohjelmointilogiikkaa.

5.3 Sovelluksen koodin kuvaus

Toteutettava sovellus on yksinkertainen mutta käytännöllinen ”To-Do” tehtäväluettelosovellus, jossa käyttäjä voi lisätä tehtäviä ja poistaa niitä klikkaamalla. Harjoitus tehdään askel askeleelta, ja opiskelijan tarvitsee vain luoda tyhjä QML-projekti Qt Creatorilla ja muokata main.qml-tiedostoa. Tehtävässä tutustutaan näin deklaratiiviseen käyttöliittymäsuunnitteluun, eikä C++-ohjelmointia tarvita.

QML on suunniteltu deklaratiiviseksi kielimalliksi, mikä tarkoittaa, että käyttöliittymä ja sen rakenne määritellään koodissa korkealla tasolla ilman tarvetta rakentaa toteutuksen erilaisia komponentteja ohjelmallisesti. Tämä ilmenee heti harjoituksen ensimmäisessä vaiheessa, jossa ApplicationWindow, TextField, Button ja ColumnLayout asetetaan selkeästi päällekkäin käyttöliittymän rakenteeksi. Jokainen komponentti määritellään suoraan osaksi hierarkiaa, ja niiden ulkoasu sekä sijoittelu toteutetaan layout-määrityksillä kuten Layout.fillWidth ja anchors.margins.

Deklaratiivinen tyyli tekee koodista luettavaa ja visuaalisesti helposti hahmotettavaa, erityisesti käyttöliittymien tapauksessa. Esimerkiksi kun painikkeen enabled-tila on sidottu suoraan tekstikentän text.length-arvoon, ei tarvita erillistä funktiota tai tilamuuttujaa seuraamaan kentän sisältöä. Tämä helpottaa käyttöliittymän suunnittelua ja ylläpitoa.

Signaalit ovat yksi Qt:n kulmakivistä, ja QML mahdollistaa niiden helpon käytön. Harjoituksessa signaalien käyttö näkyy useassa kohdassa: erityisesti Button-komponentin onClicked-lohko reagoi painikkeen clicked-signaaliin. Tämä signaali käynnistää tehtävän lisäämisen ListModeliin ja tyhjentää tekstikentän.

Toisena esimerkkinä on MouseArea-komponentti tehtäväkorttien sisällä, jonka onClicked-signaali reagoi käyttäjän napsautukseen ja käynnistää poistoprosessin.

Edistyneemmässä muodossa signaalien käsittely laajennetaan myöhemmässä vaiheessa deleteMe-signaaliksi, joka määritellään tehtäväkortin sisällä ja yhdistetään Connections-rakenteella taskModel.remove(index)-kutsuun. Tämä erottaa käyttöliittymälogiikan datalogiikasta (mallin muokkaus) selkeästi ja osoittaa, miten signaaleilla voidaan rakentaa reaktiivinen, vuorovaikutteinen käyttöliittymä.

QML:ssä on laaja valikoima valmiita käyttöliittymäkomponentteja, erityisesti QtQuick.Controls-moduulin kautta. Harjoituksessa käytetään mm. TextField, Button, ListModel, GridView, Rectangle, Text, MouseArea ja Layout-komponentteja. Nämä komponentit tarjoavat runsaasti valmiita ominaisuuksia, kuten tekstin syöttö, hiiren klikkausten käsittely, mallien esittäminen visuaalisesti sekä automaattinen asettelu.

Käyttämällä näitä valmiita komponentteja sovelluksen käyttöliittymä säilyy yhtenäisenä ja skaalautuvana, samalla kun kehittäjän työ helpottuu. Erityisesti GridView on hyvä esimerkki komponentista, joka hoitaa suuren osan tehtävien asettelusta ikkunaan ilman tarvetta koordinaattien käsittelylle.

Viimeisessä vaiheessa harjoitusta tuodaan mukaan QML:n vahva animaatiotuki. Esimerkiksi komponentti SequentialAnimation näyttää miten käyttöliittymäelementtien siirtymät voidaan määritellä visuaalisesti miellyttäviksi ilman, että erikseen tarvitsee ohjelmoida aikataulutuslogiikkaa. Kun uusi tehtävä lisätään listaan, se ilmestyy fade-in-animaationa, ja poistettaessa se häviää fade-out-efektillä ennen kuin se poistetaan mallista.

Animaatiot on toteutettu täysin deklaratiivisesti: niitä ei ajeta manuaalisesti vaan ne tapahtuvat automaattisesti, kun ominaisuuden arvo muuttuu. Tämä lähestymistapa tekee animaatioiden lisäämisestä erittäin helppoa ja mahdollistaa näyttävän käyttökokemuksen ilman raskasta animaatioiden toimintalogiikoiden

koodaamista käsin. QML tarjoaa valmiit työkalut visuaalisesti sulavien käyttöliittymien rakentamiseen, ja tämä vaihe tuo sen selkeästi esiin.

```
import QtQuick
import QtQuick.Controls
import QtQuick.Layouts

ApplicationWindow {
    visible: true
    width: 400
    height: 600
    title: "To-Do List"

    ListModel {
        id: taskModel
    }

    ColumnLayout {
        anchors.fill: parent
        anchors.margins: 20
        spacing: 10

        TextField {
            id: taskInput
            placeholderText: "Enter a task..."
            Layout.fillWidth: true
        }

        Button {
            text: "Add Task"
            Layout.fillWidth: true
            enabled: taskInput.text.length > 0
            onClicked: {
                taskModel.append({ name: taskInput.text })
                taskInput.text = ""
            }
        }

        GridView {
            model: taskModel
            cellWidth: 130
            cellHeight: 70
            Layout.fillWidth: true
            Layout.fillHeight: true
            Layout.alignment: Qt.AlignHCenter

            delegate: Rectangle {
                id: taskCard
                width: 120
                height: 60
                color: "#d0f0c0"
                radius: 10
                border.color: "gray"
                opacity: 0.0

                signal deleteMe()

                Text {
                    anchors.centerIn: parent
                    text: name
                    wrapMode: Text.Wrap
                    horizontalAlignment: Text.AlignHCenter
                }
```

```
                Behavior on opacity {
                    NumberAnimation { duration: 200 }
                }

                Component.onCompleted: opacity = 1.0

                MouseArea {
                    anchors.fill: parent
                    onClicked: {
                        fadeOut.start()
                    }
                }

                SequentialAnimation {
                    id: fadeOut
                    PropertyAnimation {
                        target: taskCard
                        property: "opacity"
                        to: 0.0
                        duration: 200
                    }
                    ScriptAction {
                        script: taskCard.deleteMe()
                    }
                }

                Connections {
                    target: taskCard
                    function onDeleteMe() {
                        taskModel.remove(index)
                    }
                }
            }

        }
    }
}
```

Esimerkkikoodi 3. Harjoituksen lopputuloksena syntyvä koodi.

5.4 Lopputulos

Tämä summatiivinen harjoitus tarjoaa mahdollisuuden soveltaa ja syventää Qt Quick -osaamista. Harjoituksen koodissa tutustutaan deklaratiiviseen käyttöliittymäsuunnitteluun, dynaamiseen tietojen sidontaan, valmiiden komponenttien hyödyntämiseen ja animaatioiden lisäämiseen. Qt:n vahvuudet, kuten selkeys, käytön helppous ja visuaalinen vaikuttavuus, tulevat harjoituksessa esiin, antaen kattavan kuvan Qt:n mahdollisuuksista käyttöliittymäsovellusten kehittämisessä.

6 Yhteenveto

Tämä insinöörityö käsittelee summatiivisen ohjelmointiharjoituksen kehittämistä Qt-ohjelmoinnin verkko-opintojaksolle. Työssä tarkastellaan erityisesti Qt-sovelluskehyksen ominaisuuksia ja vertaillaan niitä Electron- ja GTK-kehysten ominaisuuksiin. Qt:n vahvuuksia, kuten erinomainen suorituskyky, natiivien käyttöliittymien toteutus ja monipuoliset kehitystyökalut (erityisesti Qt Creator), käsitellään yksityiskohtaisesti. Työssä esitetään Qt:n käyttöliittymäkehityksen perusteita QML:n ja C++:n avulla, esimerkkikoodien kautta.

Vertailussa Electron osoittautuu web-kehittäjille helpoksi lähestyttäväksi, mutta sen resurssikulutus on suuri verrattuna Qt:hen ja GTK:hon. GTK tarjoaa hyvän suorituskyvyn ja tehokkaan alustan erityisesti Linux-ympäristöissä, mutta sen käyttö Windowsissa ja macOS:ssa on rajatumpaa verrattuna Qt:hen. Qt osoittautuu tehokkaimmaksi ratkaisuksi monimutkaisissa ja resursseja vaativissa sovelluksissa, erityisesti sulautetuissa järjestelmissä ja teollisuussovelluksissa.

Osana työtä kehitettiin QML-pohjainen ”To-Do” -sovellusharjoitus, joka opettaa keskeiset Qt Quick -ominaisuudet, kuten deklaratiivisen ohjelmointityylin, signaalien hyödyntämisen, valmiiden komponenttien käytön sekä animaatioiden lisäämisen. Tämä harjoitus demonstroi Qt:n vahvuuksia helppokäyttöisenä ja visuaalisesti vaikuttavana käyttöliittymäkehityksen työkaluna.

Työn loppupäätelmänä todetaan Qt:n soveltuvan erityisesti projekteihin, joissa korostuvat suorituskyky, monialustaisuus ja natiivin käyttökokemuksen merkitys. Electron ja GTK voivat olla sopivia vaihtoehtoja tilanteisiin, joissa joko kehittäjän osaaminen tai alustan erityistarpeet korostuvat.

6.1 Pohdinta

Itselleni työssä korostui eri kehitystyökalujen merkitys käytännön ohjelmoinnissa. Totesin Qt Creatorin tehokkaaksi ympäristöksi, joka tarjoaa monipuoliset työkalut sekä käyttöliittymien visuaaliseen suunnitteluun että sovelluslogiikan

debuggaamiseen. GTK:n ja Electronin osalta työkalujen hajanaisuus sekä erilaiset kielivaatimukset nousivat esille haastavina tekijöinä.

Kokeillessa käyttöliittymien rakentamista korostui minulle myös yleisen visuaalisen houkuttelevuuden ja käyttökokemuksen merkitys nykyaikaisissa sovelluksissa. Qt Quickin valmiudet luoda dynaamisia, animaatioita hyödyntäviä käyttöliittymiä ilman suurta ohjelmointikuormaa ja ennen kaikkea kyky helposti kokeilla asioita tekee siitä erittäin hyvän vaihtoehdon erityisesti mobiili- ja sulautettujen järjestelmien sovelluksissa, joissa ruututilaa ei ole paljoa ja pitää keskittyä olennaiseen.

Kokonaisuutena koen, että Qt tarjoaa vahvan pohjan tulevaisuuden tarpeisiin sovellusten toteuttamisessa, samalla kun Electron ja GTK pysyvät merkittävinä vaihtoehtoina tietyissä käyttötapauksissa, joissa niiden omat vahvuudet tulevat esille.

Lähteet

1 Qt Group. Qt History. 2024. https://wiki.qt.io/Qt_History Luettu 20.5.2025

2 Qt Group. Qt Quick and Widgets, Qt 6.4 Edition. 2022. https://www.qt.io/blog/qt-quick-and-widgets-qt-6.4-edition

3 Qt Group. Qt Widgets. 2025. https://doc.qt.io/qt-6/qtwidgets-index.html

4 Qt Group. Qt Creator Overview. 2025. https://doc.qt.io/qtcreator/creator-overview.html

5 Qt Group. Qt Linguist Manual. 2025. https://doc.qt.io/qt-6/qtlinguist-index.html

6 Qt Group. List of Qt Applications. 2025. https://wiki.qt.io/List_of_Qt_Applications

7 Scythe Studio. QML vs Qt Widgets: detailed comparison. 2025. https://scythe-studio.com/en/blog/qml-vs-qt-widgets-detailed-comparison Luettu 25.5.2025.

8 OpenJS Foundation. Atom Shell is now Electron. 2015. https://www.electronjs.org/blog/electron

9 OpenJS Foundation. Electron Docs: Why Electron. 2025. https://www.electronjs.org/docs/latest/why-electron Luettu 20.5.2025

10 Medium. Quick look: Electron vs Qt/QML app memory usage. 2025. https://pkoretic.medium.com/quick-look-electron-vs-qt-qml-app-memory-usage-e8769008534f Luettu 5.6.2025.

11 GTK Team. The GTK Project. 2025. https://www.gtk.org

12 Medium. Comparing Desktop Application Development Frameworks: Electron, Flutter, Tauri, React Native, and Qt. 2023. https://medium.com/@maxel333/comparing-desktop-application-development-frameworks-electron-flutter-tauri-react-native-and-fd2712765377

13 GTK Development Blog. New renderers for GTK. 2024. https://blog.gtk.org/2024/01/28/new-renderers-for-gtk

14 Test Mace. Why we have chosen Electron. 2019. https://testmace.com/blog/2019/01/15/why-electron/ Luettu 20.5.2025

15 GTK Docs. Overview of GTK and its Libraries. 2025. https://gtk.org/docs/architecture/