

QT QML

Radosavljević Milan 1434

Napredno softversko inženjerstvo

Mentor: Aleksandar Milosavljević

Sadržaj

01

Opšte o QT okviru

...

02

QtQuick

...

03

Integracija QML-a i C++-a

...

04

Projekat Insight


...



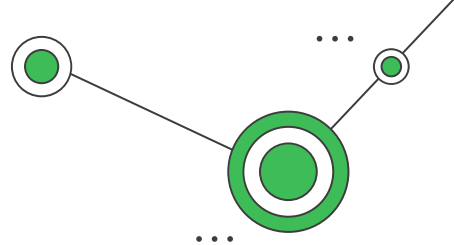


01

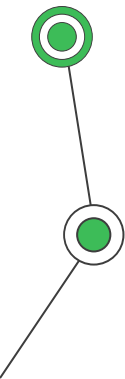
Opšte o Qt okviru



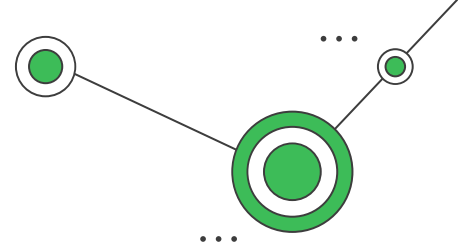
Qt



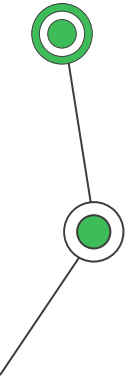
- Qt (izgovara se "kjut") je okvir za razvoj grafičkih korisničkih interfejsa kao i aplikacija koje imaju podršku za različite platforme kao što su Linux, Windows, macOS, Android i pruža podršku za različite hardverske platforme i **embedded** sisteme.
- Qt pruža mogućnost da bez izmena osnovnog koda aplikacije imaju izvornu podršku i brzinu izvršenja na svim podržanim platformama.



Qt



- Qt se trenutno razvija od The Qt Company i od Qt Project kao projekat otvorenog koda, uključujući individualce i samu organizaciju koja radi na unapređenju Qt okvira.
- Qt je dostupan i sa vlasničkim licencama i sa licencom otvorenog koda.
- Poslednja verzija okvira je Qt 6.1.



Glavni koncepti Qt okvira



QObject

QObject je roditeljska klasa gotovo svih Qt klasa i svih klasa za kontrole.

...



Signali i slotovi

Signali i slotovi su mehanizam koji se koristi za komunikaciju između objekata.

...



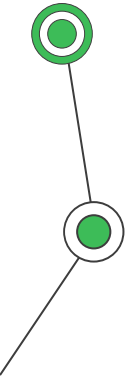
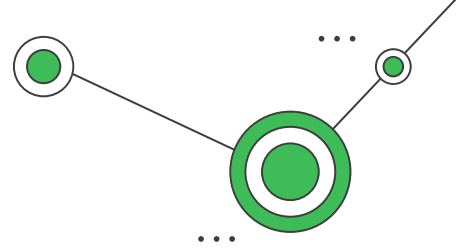
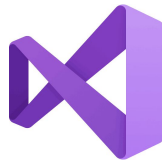
MOC

Omogućava meta-podatke, mehanizam signala i slotova i generiše dodatne .cpp i .h fajlove.

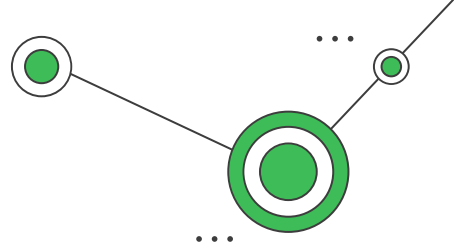
...

Okruženja za razvoj Qt aplikacija

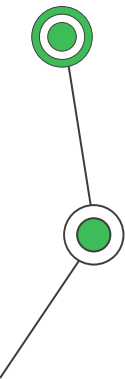
- Qt Creator
- Visual Studio
- KDevelop



Zašto Qt?



- Pruža veliki broj mogućnosti (npr. rad sa XML-om, mrežama, OpenGL-om, nitima..), pritom okvir održava isti pristup rada
- Mogućnost *portovanja* na različite platforme
- Perfektna dokumentacija
- Najpribližniji Javi i .NET-u po pitanju produktivnosti

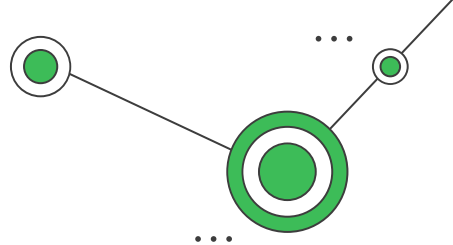


02

QtQuick

QML

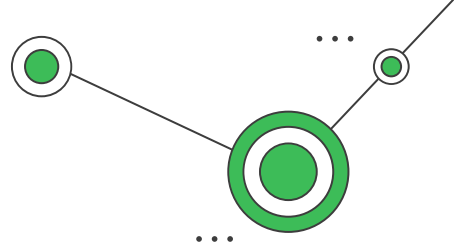
QtQuick



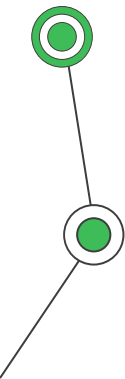
- GUI je moguće razvijati kroz forme, a takođe je moguće brzo razviti korisnički interfejs korišćenjem QML-a
- QML je deklarativni jezik opisa objekata koji integriše Javascript za proceduralno programiranje
- QtQuick pruža neophodne module za razvoj GUI-a pomoću QML-a. Moguće je pisati celu aplikaciju samo u QML-u, ali obično je samo GUI napisan u QML-u, a pozadina aplikacije se implementira u C++



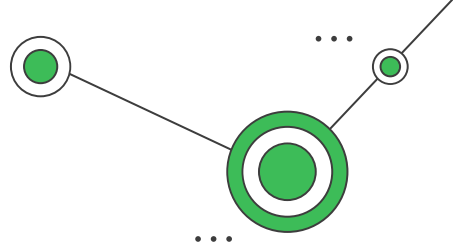
QML



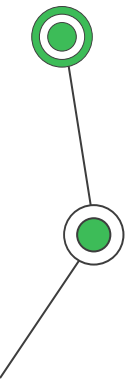
- QML je deklarativni jezik koji omogućava korisničkim interfejsima da budu opisani u pogledu njihovih vizuelnih komponenti i kako oni uzajamno deluju i odnose se jedni sa drugima.
- To je visoko čitljiv jezik koji je dizajniran da omogućí komponentama da budu međusobno povezani na dinamičan način, i omogućava komponentama da se lako ponovo koriste i prilagođavaju unutar korisničkog interfejsa.
- Koristeći QtQuick modul, dizajneri i programeri mogu lako da izgrade fluidno-animirane korisničke interfejse u QML-u, i imaju mogućnost povezivanja ovih korisničkih interfejsa sa bilo kojim bibliotekama u C++.



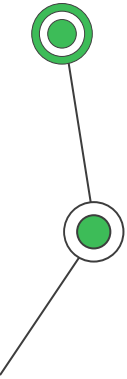
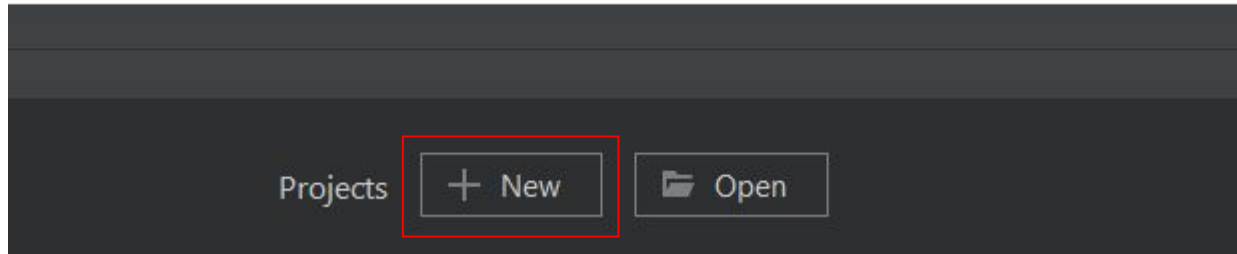
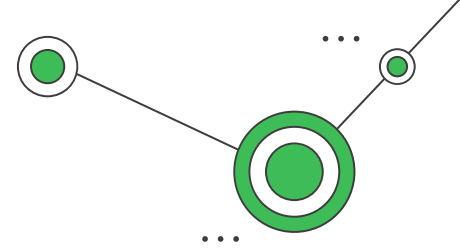
QtQuick vs Qt Widgets



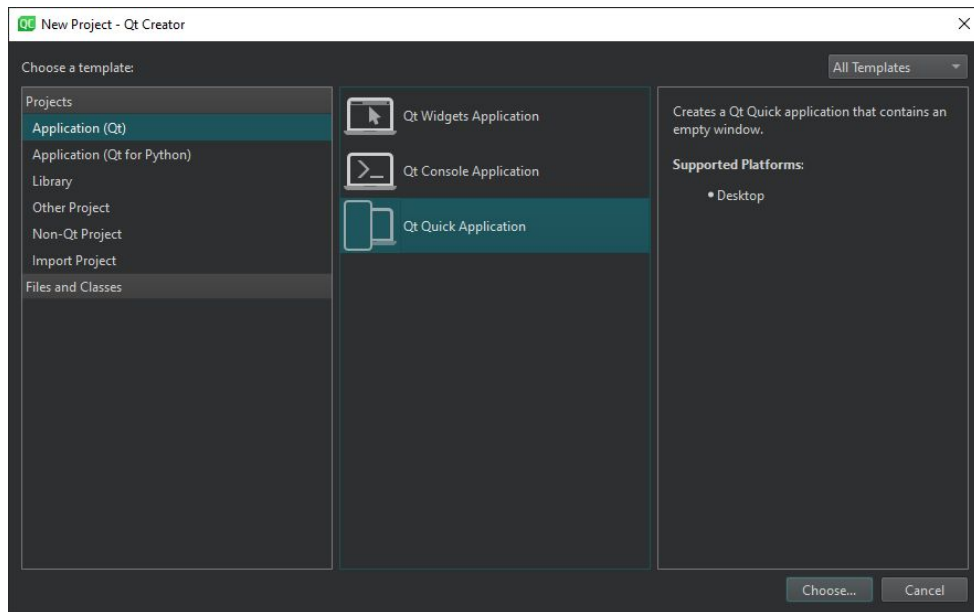
- QtQuick pruža fleksibilniji pristup.
- Brži razvoj GUI-ja u QtQuicku.
- QtQuick omogućava razne mogućnosti pozicioniranja (npr. usidranje), dok Qt Widgets omogućava samo rasporede.
- QtQuick ima lošije performanse u poređenju sa Qt Widget pristupom.



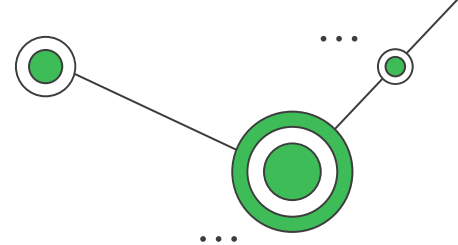
Kreiranje QtQuick projekta



Kreiranje QtQuick projekta



Kreiranje QtQuick projekta



Qt Quick Application - Qt Creator

Project Location

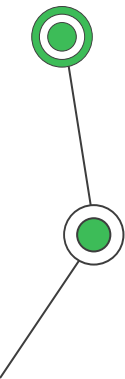
Creates a Qt Quick application that contains an empty window.

Location
Build System
Details
Translation
Kits
Summary

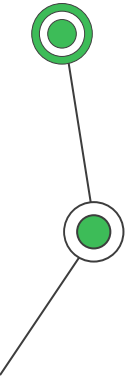
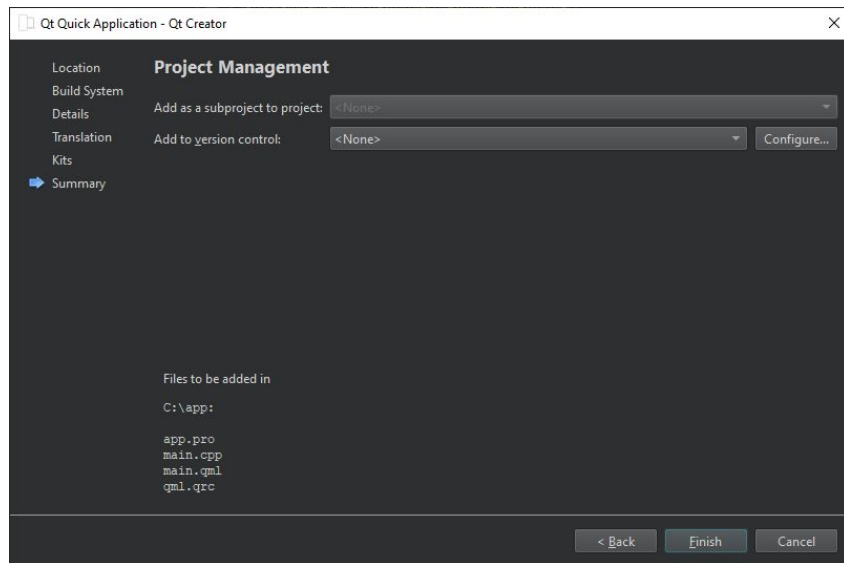
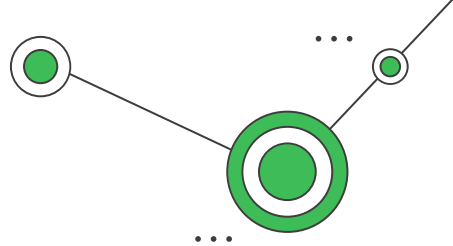
Name:

Create in:

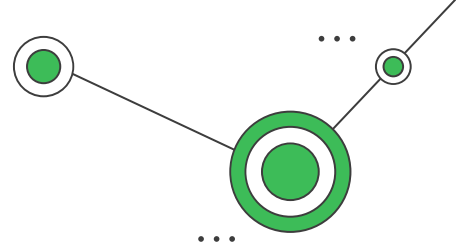
☐ Use as default project location



Kreiranje QtQuick projekta



QML Sistem tipova



Tipovi koji se koriste u definisanju hierarhije QML objekata mogu da se omoguće iz različitih izvora.

- Omogućeni u samom QML jeziku
- Registrovani kroz C++
- Omogućeni kao QML dokumenti

Korisnici mogu da omoguće tipove registrovanjem C++ tipova direktno ili mogu da definišu komponentu koja se može koristiti više puta i koja se može integrisati u projekat.

Bez obzira odakle dolazi definisanje tipa, mehanizam će sprovesti type-safety proceduru za svojstva i instance tih tipova.



Osnovni QML tipovi

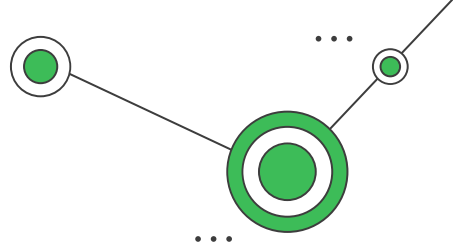
Osnovni tipovi su podrazumevano podržani od samog mehanizma i ne zahtevaju *importovanje* modula da bi bili korišćeni.

- Bool - binarni tip
- Double - realni tip sa dvostrukom preciznošću
- Enumeration - tipovi nabiranja
- Int - Celobrojni tip
- List - Lista QML objekata
- Real - Realni tip
- String - Tekst
- Url - Lokator resursa
- Var - Generički tip svojstva

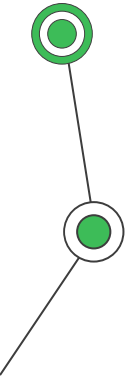
Dok drugi tipovi zahtevaju od klijenta da eksplicitno *importuje* module kako bi obezbedio tipove.

- Date - Datum
- Point - Vrednost sa x, y koordinatom
- Rect - Vrednost sa x, y koordinatom, visinom i širinom
- Size - Vrednost sa visinom i širinom

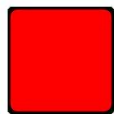
Component tip



- Enkapsulira definiciju QML komponente.
- Praktično predstavlja osnovni tip u QML-u za prikaz.
- Komponente se često koriste za definiciju komponentnih datoteka, tačnije *.qml* datoteka.



Komponente



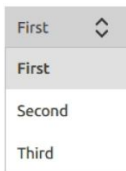
Rectangle

Iscrtava se popunjeni pravougaonik.



Button

Iscrtava se dugme koje može da obradi klik.



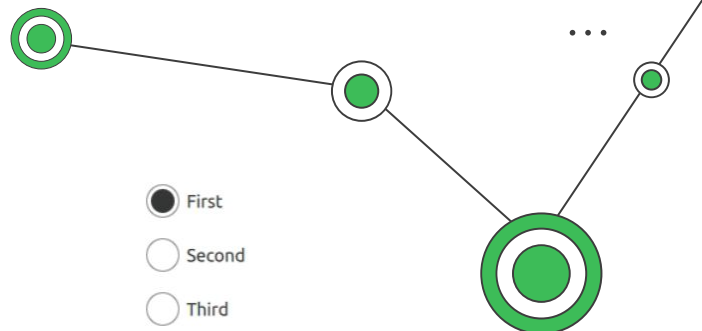
ComboBox

Kombinacija dugmeta i popup liste za odabir opcija.

Hello World!

TextField

Polje koje omogućava korisniku da unese tekst.



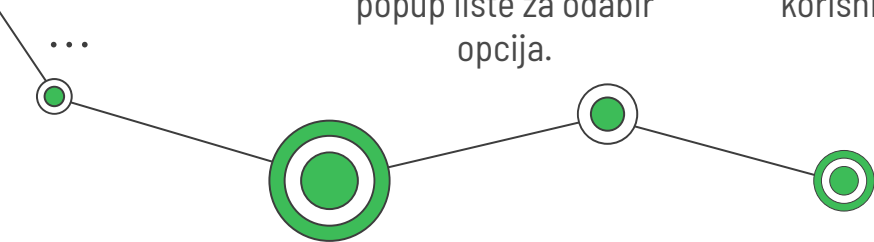
RadioButton

Ekskluzivno radio dugme.

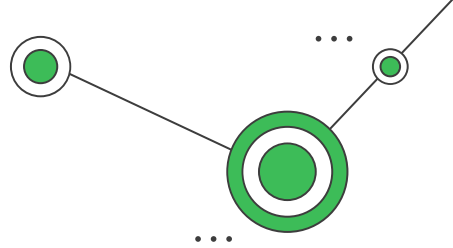


Slider

Koristi se za selekciju vrednosti pomeranjem klizača.



Rasporedi



Rasporedi su komponente koje se koriste za uređivanje drugih komponenti u korisničkom interfejsu. S obzirom da rasporedi i menjaju veličinu komponenti, predstavljaju odličan izbor za korisničke interfejse sa promenjivom veličinom. Moguće je podešavati:

- **Poravnanje** - Moguće je specificirati poravnanje u okviru ćelije koje će komponenta da okupira.
- **Promena veličine** - Moguće je specificirati da li će komponenta da popuni ćeliju po visini i/ili širini
- **Ograničenja veličina** - Moguće je specificirati minimalnu, maksimalnu i preferiranu visinu ili širinu.
- **Razmaci** - Moguće je specificirati razmake između ćelija.

Tipovi rasporeda:

- Raspored po principu rešetke (**GridLayout**) - Omogućava dinamičko raspoređivanje komponenti u rešetku
- Raspored po redu (**RowLayout**) - Identičan kao Raspored po principu rešetke, samo što ima jedan red
- Raspored po koloni (**ColumnLayout**) - Identičan kao Raspored po principu rešetke, samo što ima jednu kolonu
- Raspored po principu magacina (**StackLayout**) - Ovaj raspored omogućava da se samo jedna komponenta vidi u trenutku.



Rasporedi – Primer

```
GridLayout {
    id: grid
    columns: 3

    Text { text: "Three"; font.bold: true; }
    Text { text: "words"; color: "red" }
    Text { text: "in"; font.underline: true }
    Text { text: "a"; font.pixelSize: 20 }
    Text { text: "row"; font.strikeout: true }
}
```

Three words in
a row

```
ColumnLayout {
    spacing: 2

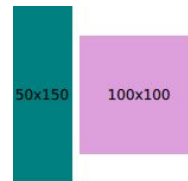
    Rectangle {
        Layout.alignment: Qt.AlignCenter
        color: "red"
        Layout.preferredWidth: 40
        Layout.preferredHeight: 40
    }

    Rectangle {
        Layout.alignment: Qt.AlignRight
        color: "green"
        Layout.preferredWidth: 40
        Layout.preferredHeight: 70
    }

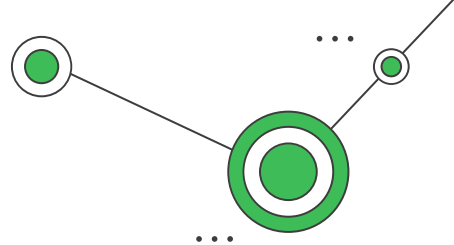
    Rectangle {
        Layout.alignment: Qt.AlignBottom
        Layout.fillHeight: true
        color: "blue"
        Layout.preferredWidth: 70
        Layout.preferredHeight: 40
    }
}
```



```
RowLayout {
    id: layout
    anchors.fill: parent
    spacing: 6
    Rectangle {
        color: 'teal'
        Layout.fillWidth: true
        Layout.minimumWidth: 50
        Layout.preferredWidth: 100
        Layout.maximumWidth: 300
        Layout.minimumHeight: 150
        Text {
            anchors.centerIn: parent
            text: parent.width + 'x' + parent.height
        }
    }
    Rectangle {
        color: 'plum'
        Layout.fillWidth: true
        Layout.minimumWidth: 100
        Layout.preferredWidth: 200
        Layout.preferredHeight: 100
        Text {
            anchors.centerIn: parent
            text: parent.width + 'x' + parent.height
        }
    }
}
```

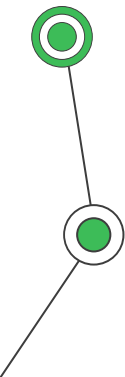
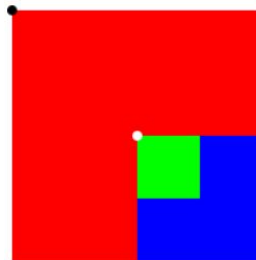


Prostor za iscrtavanje

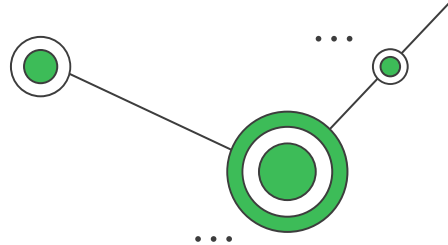


- Koordinatni sistem
 - Pikel gore-levo predstavlja (0,0) tačku u QtQuick koordinatnom sistemu. Koordinatni sistem vizuelnog potomka je relativan u odnosu na njegovog vizuelnog roditelja.
- Sistem roditelja
 - U QtQuick-u postoje dve različite vrste roditelja. Prvi kao roditelj posedovanja i drugi kao vizuelni roditelj, koji određuje gde se na ekranu iscrtava komponenta.

```
Rectangle {  
    width: 200  
    height: 200  
    color: "red"  
  
    Rectangle {  
        x: 100  
        y: 100  
        width: 100  
        height: 100  
        color: "blue"  
  
        Rectangle {  
            width: 50  
            height: 50  
            color: "green"  
        }  
    }  
}
```



Interakcija sa korisnikom

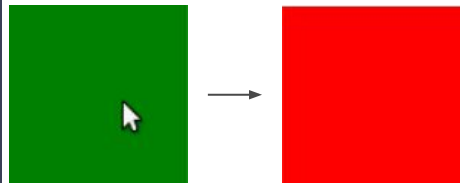


- Obrada događaja miša
 - MouseArea je nevidljiva kontrola koja se koristi u kombinaciji sa vidljivom kontrolom za postizanje obrade događaja miša na vidljivoj kontroli.

```
import QtQuick 2.0

Rectangle {
    width: 100; height: 100
    color: "green"

    MouseArea {
        anchors.fill: parent
        onClicked: { parent.color = 'red' }
    }
}
```



- Obrada tastera
 - Sve visualne primitive podržavaju obradu dugmića kroz dodavanje Keys svojsta. Moguće je obraditi pritisak tastera pri pritisku ili pri njegovom otpuštanju.

```
Item {
    anchors.fill: parent
    focus: true
    Keys.onPressed: {
        if (event.key == Qt.Key_Left) {
            console.log("move left");
            event.accepted = true;
        }
    }
}
```


Pozicioniranje



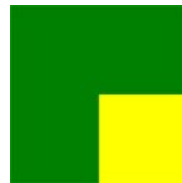
- Ručno pozicioniranje
 - Ako je korisnički interfejs statičan, ručno pozicioniranje predstavlja najefikasniji način za pozicioniranje. Što znači eksplicitno navođenje koordinata pozicioniranja.
- Pozicioniranje izrazima
 - Komponente takođe mogu da budu pozicionirane vezivanjem izraza za svojstva pozicioniranja. Ovaj način pozicioniranja veoma utiče na performanse.
- Usidravanje
 - Sidra (anchor) omogućavaju da se komponenta postavi pored ili unutar druge, pričvršćivanjem jedne ili više sidrišnih tačaka komponente na sidrenu tačku druge. Tačke usidravanja će ostati čak i ako se promene dimenzije ili lokacija jedne od stavki.

```
import QtQuick 2.0
```

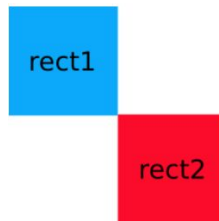
```
Item {  
    width: 200  
    height: 200
```

```
    Rectangle {  
        x: 50  
        y: 50  
        width: 100  
        height: 100  
        color: "green"  
    }
```

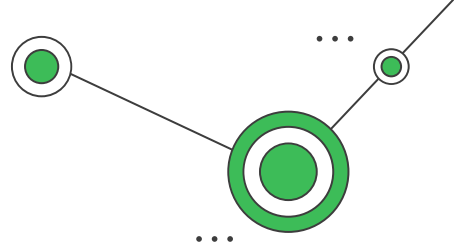
```
    Rectangle {  
        x: 100  
        y: 100  
        width: 50  
        height: 50  
        color: "yellow"  
    }  
}
```



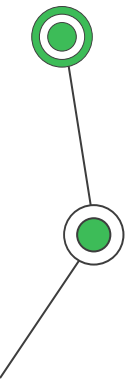
```
Rectangle { id: rect1; ... }  
Rectangle { id: rect2; anchors.left: rect1.right; anchors.top: rect1.bottom; ... }
```



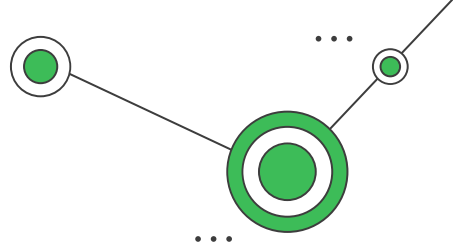
Stanja



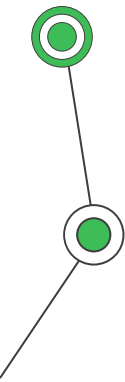
- Stanje određene vizuelne komponente je skup informacija koje opisuju kako i gde su pojedinačni sastavni delovi vizuelne komponente prikazani unutar nje, kao i svi podaci povezani sa tim stanjem.
- Većina vizuelnih komponenti u korisničkom interfejsu će imati ograničen broj stanja, svako sa dobro definisanim svojstvima.
- Mnogi dizajni korisničkog interfejsa su vođeni stanjima, tačnije interfejsi imaju konfiguracije koje se razlikuju u zavisnosti od trenutnog stanja.



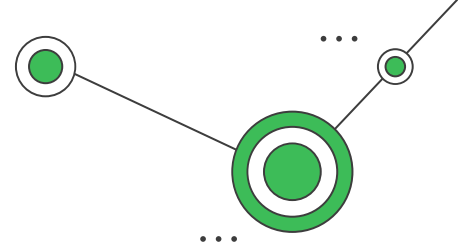
Stanja



- U QML-u, stanja su skup konfigurisanih svojstava definisanih u nekom stanju.
- Svi objekti zasnovani na komponentama imaju svojstvo stanja i mogu specificirati dodatna stanja dodavanjem novih *State* objekata.
- Svako stanje unutar komponente ima jedinstveno ime, a stanje čije ime je prazan string je podrazumevano.
- Da bi promenili trenutno stanje stavke, potrebno je da se svojstvo *state* postavi na ime željenog stanja.



Stanja

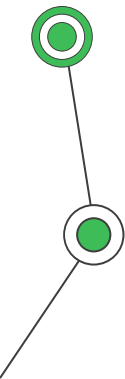


```
Rectangle {  
  width: 75; height: 75  
  id: button  
  state: "RELEASED"  
  
  MouseArea {  
    anchors.fill: parent  
    onPressed: button.state = "PRESSED"  
    onReleased: button.state = "RELEASED"  
  }  
  
  states: [  
    State {  
      name: "PRESSED"  
      PropertyChanges { target: button; color: "lightblue"}  
    },  
    State {  
      name: "RELEASED"  
      PropertyChanges { target: button; color: "lightsteelblue"}  
    }  
  ]  
}
```

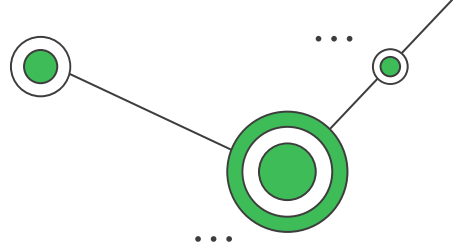
← Inicijalno stanje

← Promena stanja

← Definisanje stanja

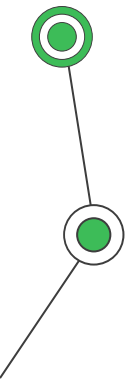


Tranzicije ⇄

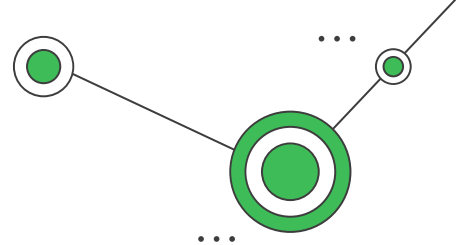


Kada vizuelna komponenta pređe iz jednog stanja u drugo, izgled te stavke će se promeniti. Tranzicija je veza između dva stanja. To može izazvati druge događaje, jer drugi delovi aplikacije mogu imati ponašanje koje se pokreće kada se uđe ili napusti određeno stanje. QtQuick obezbeđuje tip *Transition* koji ima svojstva koja definišu šta će se desiti kada se aplikacija promeni iz jednog stanja u drugo.

```
transitions: [  
  Transition {  
    from: "PRESSED"  
    to: "RELEASED"  
    ColorAnimation { target: button; duration: 100}  
  },  
  Transition {  
    from: "RELEASED"  
    to: "PRESSED"  
    ColorAnimation { target: button; duration: 100}  
  }  
]  
}
```



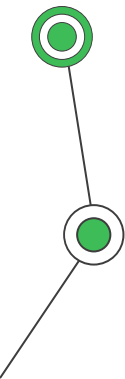
Animacije



Prilikom prelaska između stanja, fluidna animacija se može koristiti za pomoć korisniku tokom tranzicije. Nagle i neočekivane promene vizuelnog izgleda rezultuju neoptimalnim korisničkim iskustvom.

Tipovi animacija:

- `AnchorAnimation` - Animira pomeranje usidrenja
- `ColorAnimation` - Animira promenu boje
- `NumberAnimation` - Animira promenu numeričke vrednosti
- `PropertyAnimation` - Animira promenu svojstva
- `RotationAnimation` - Animacija rotacije



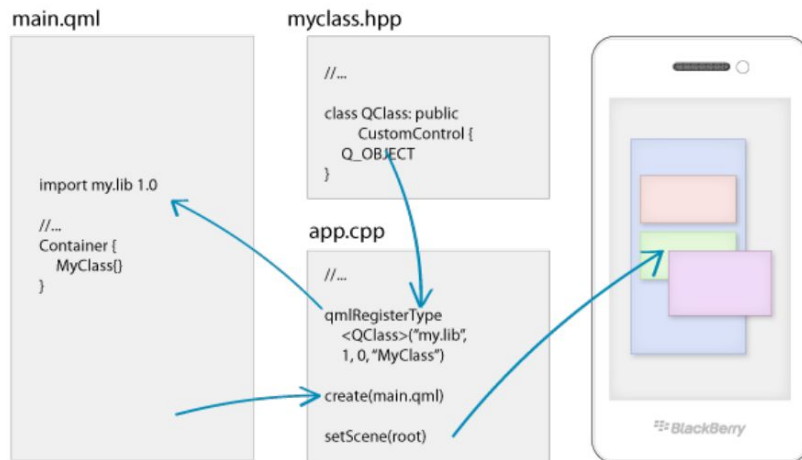
03

Integracija QML i C++

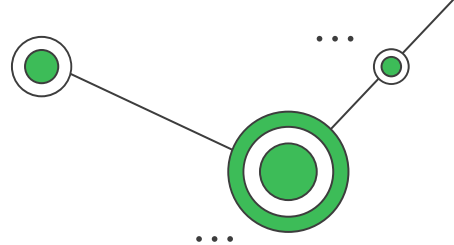
Integracija QML i C++

Aplikacije nekad moraju da izvršavaju zahtevne operacije pa je zbog toga potrebno koristiti C++. Klasa čije funkcionalnosti želimo koristiti u QML-u mora naslediti QObject klasu.

- Q_INVOKABLE - makro koji omogućava da metoda bude pozvana iz QML-a
- Q_PROPERTY - makro za dodavanje svojstva u Qt-u

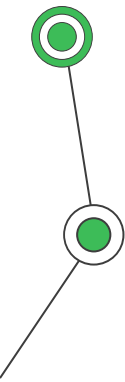
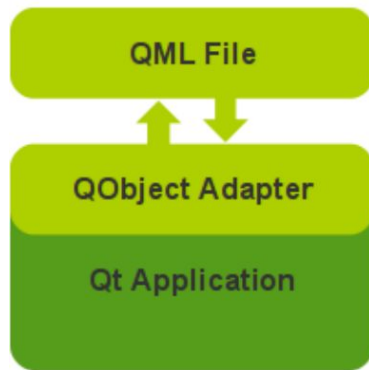


Integracija QML i C++

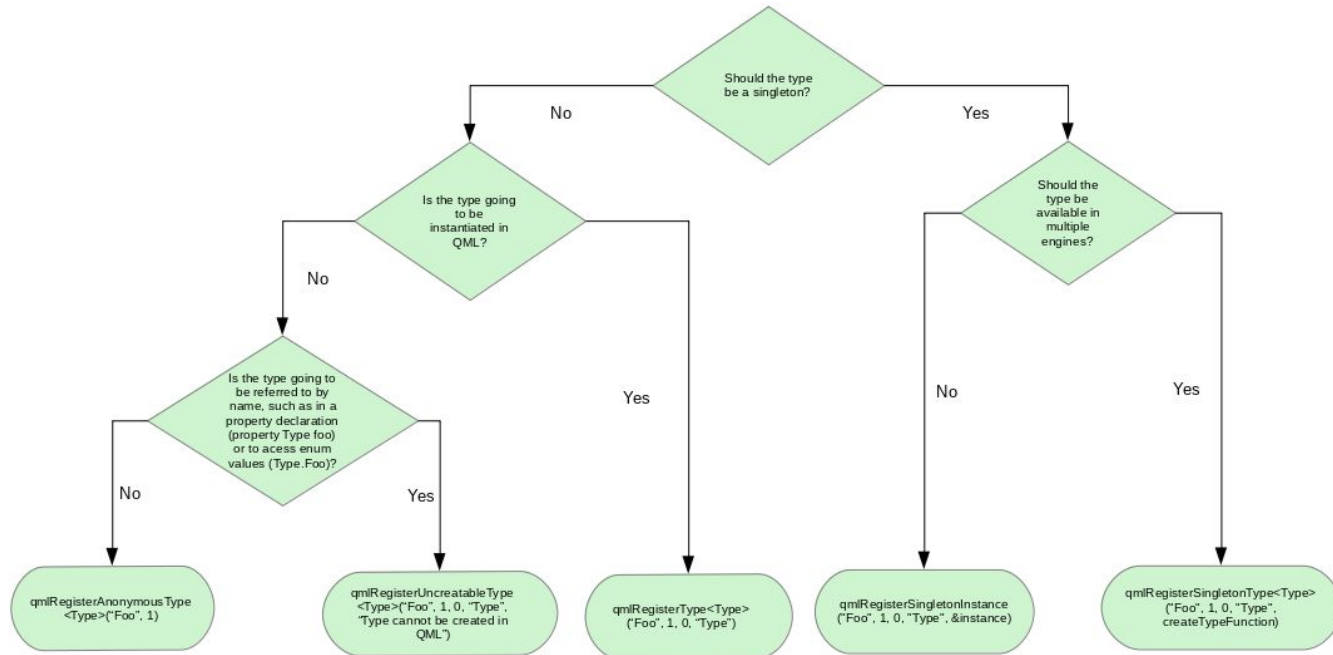


Imamo 3 načina da povežemo C++ i QML.

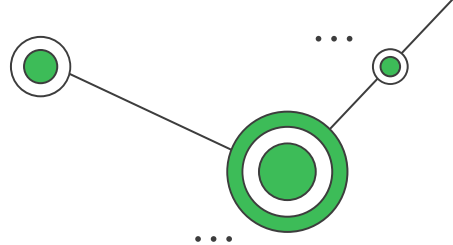
- Klasu registrujemo kao novi QML tip.
- Klasu registrujemo kao Singleton tip i tako tu instancu možemo koristiti iz QML-a.
- Kontekst promenljiva (Context property)



Integracija QML i C++



Primer registracije tipa



```
class User : public QObject
{
    Q_OBJECT
public:
    explicit User(QObject *parent = nullptr);

    Q_INVOKABLE bool isLogin() {return m_bIsLogin;}
    Q_INVOKABLE void logout() {m_bIsLogin = false;}
    Q_INVOKABLE void loginUser(QString username, QString password);
    Q_INVOKABLE void registerUser(QString username, QString nick, QString password);
    Q_INVOKABLE int getUserId() { return m_Id; }
private:
    int m_Id;
    QString m_Username;
    QString m_Nick;

    bool m_bIsLogin;

signals:
};
```

```
User user;
qmlRegisterType<User>("com.user", 1, 0, "User");
```

```
import QtQuick 2.15
import QtQuick.Window 2.15
import QtQuick.Controls 2.15
import QtQuick.Layouts 1.3
import QtPositioning 5.15
import QtLocation 5.15
import com.user 1.0
```

```
User {
    id: usermng
}
```



Primer registracije singleton tipa

```
class DatabaseManager : public QObject
{
    Q_OBJECT
private:
    explicit DatabaseManager(QObject *parent = nullptr);
    static DatabaseManager* _instance;
    QSqlDatabase db;
public:
    static DatabaseManager* instance();
    static void destroy();

public:
    bool Register(QString username, QString password, QString nickname);
    int Login(QString username, QString password);

    void InsertEvent(QString name, int type, QDateTime dateStart, QDateTime dateEnd, int createdBy, float lan, float lot);
    void UpdateEvent(int id, QString name, int type, QDateTime dateStart, QDateTime dateEnd);
    void RemoveEvent(int id);

    QString dateToString(QDateTime date);
private:
    //QML dostupne
public:
    Q_INVOKABLE QStringList getTypes();
    Q_INVOKABLE QStringList getEventsIds();
    Q_INVOKABLE QStringList getEvent(int id);

signals:
};
```

```
qmlRegisterSingletonInstance<DatabaseManager>("com.db", 1,0,"DatabaseManager", DatabaseManager::instance());
```

```
function loadItems() {
    osmMap.clearMapItems();
    var allItemsIds = DatabaseManager.getEventsIds();
```

Primer registracije tipa kao Context Property

```
class ApplicationData : public QObject
{
    Q_OBJECT
public:
    Q_INVOKABLE QDateTime getCurrentDateTime() const {
        return QDateTime::currentDateTime();
    }
};

int main(int argc, char *argv[]) {
    QApplication app(argc, argv);

    QQuickView view;

    ApplicationData data;
    view.rootContext()->setContextProperty("applicationData", &data);

    view.setSource(QUrl::fromLocalFile("MyItem.qml"));
    view.show();

    return app.exec();
}
```

```
import QtQuick 2.0

Text { text: applicationData.getCurrentDateTime() }
```



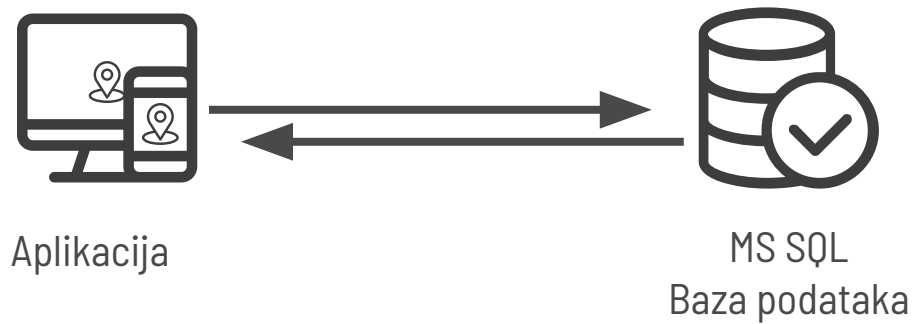
04

Projekat

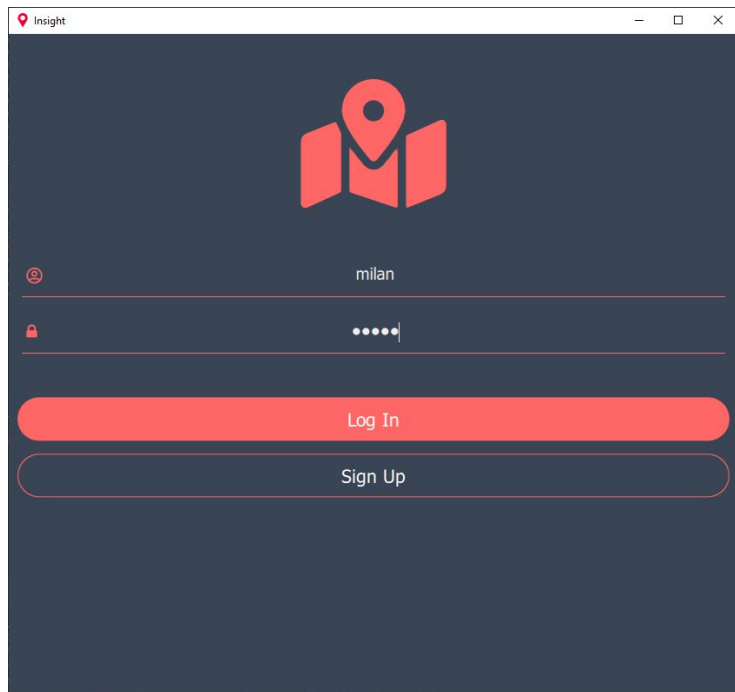
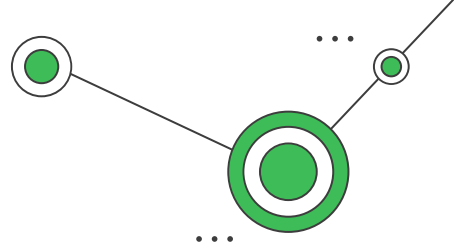
Insight




Arhitektura softvera





Sistem za upravljanje korisnicima



Insight

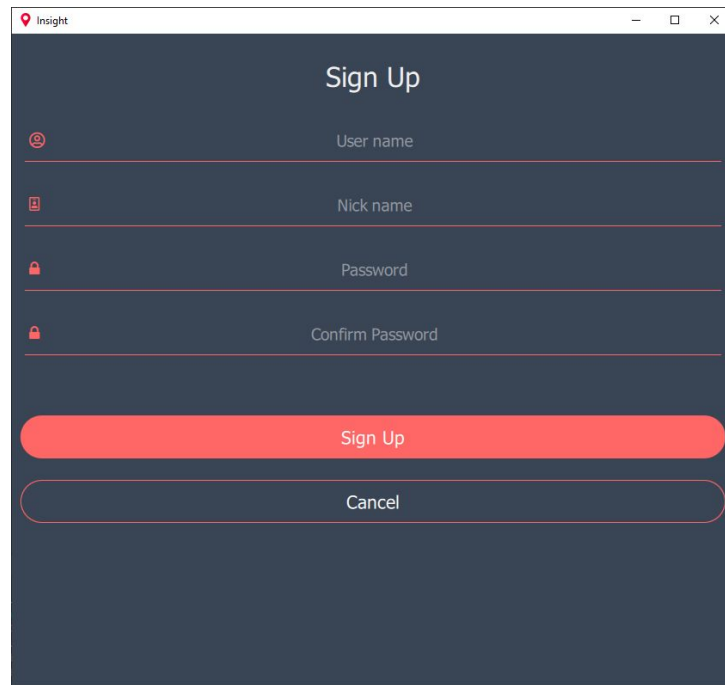


 milan




Log In


Sign Up





Insight

Sign Up

 User name

 Nick name

 Password

 Confirm Password

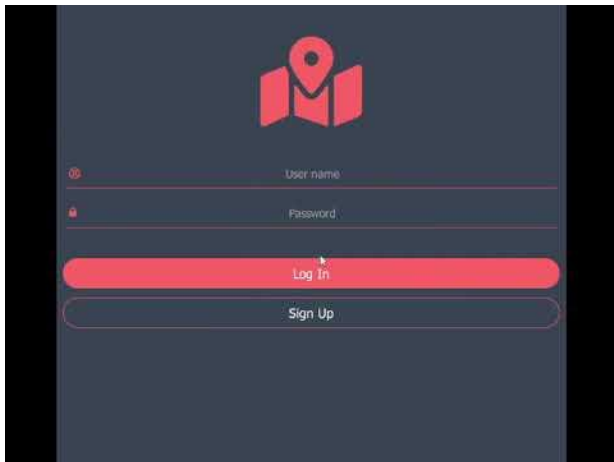
Sign Up

Cancel



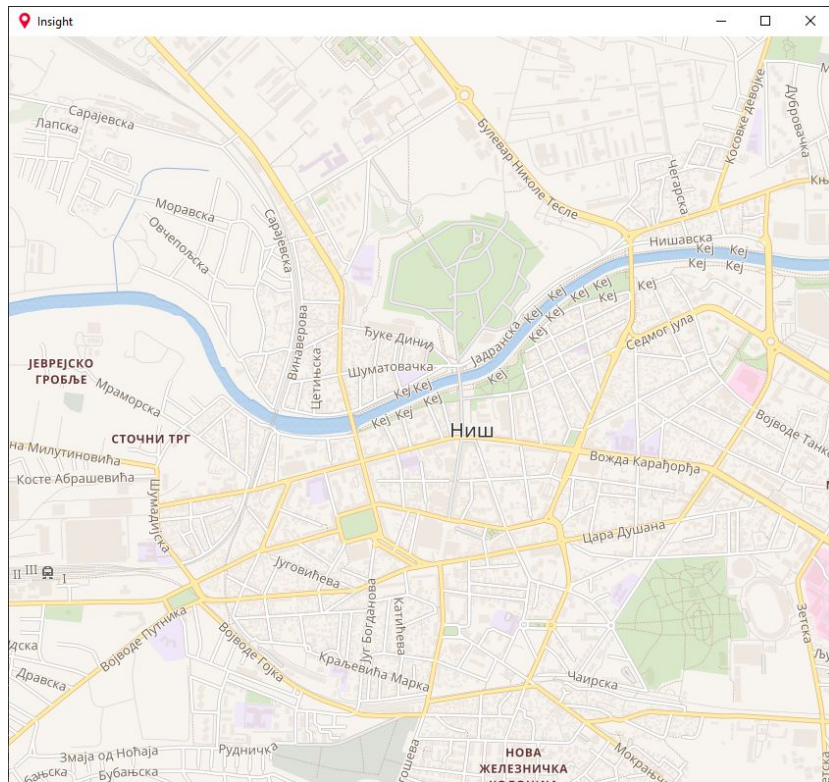
CustomButton

```
state: ""  
  
states: [  
  State {  
    name: ""  
  },  
  State {  
    name: "PRESSED"  
    PropertyChanges {  
      target: control;  
      baseColor: control.baseColor == mainAppColor ? backgroundColor : mainAppColor  
    }  
  }  
]
```



Stanja komponente su korišćena za bolje korisničko iskustvo, tačnije pri pritisku na dugme, dugme menja boju i korisnik je obavešten da je interagovao sa dugmetom.

Glavni pogled aplikacije



```
Map {
    id: osmMap
    Plugin {
        id: mapPlugin
        name: "osm" // Open Street Maps
    }

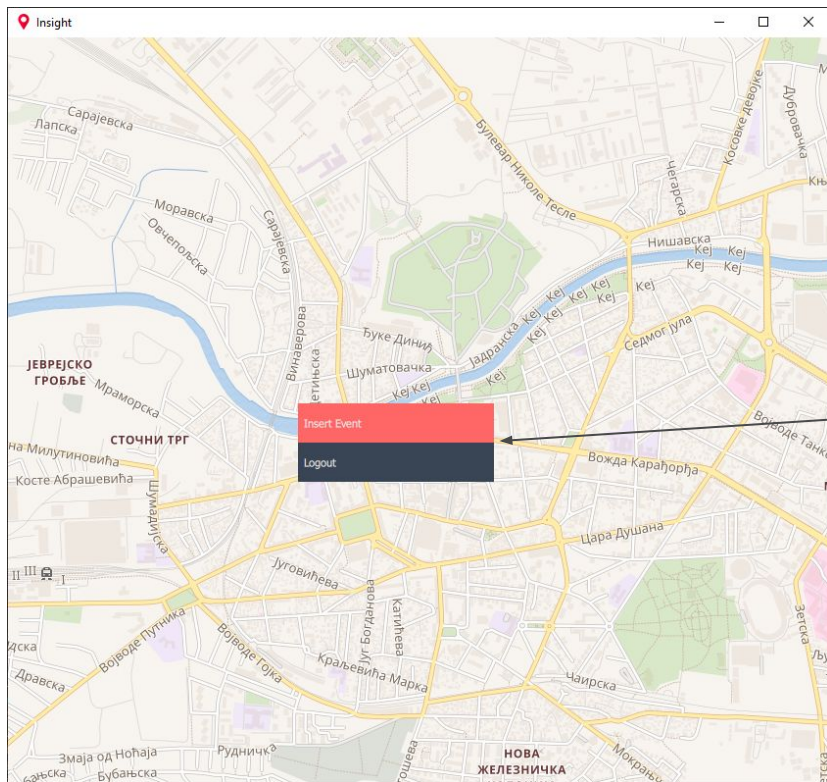
    anchors.fill: parent
    plugin: mapPlugin
    center: QtPositioning.coordinate(43.3218, 21.8937) // Nis
    zoomLevel: 15
    copyrightsVisible: false

    Component.onCompleted: {
        loadItems();
        updateTimer.start();
    }

    MouseArea {
        id: mouseArea
        anchors.fill: parent
        acceptedButtons: Qt.RightButton

        onClicked: {
            saveClickedCoord( osmMap.toCoordinate(Qt.point(mouse.x, mouse.y)) );
            contextMenu.x= mouse.x;
            contextMenu.y = mouse.y;
            contextMenu.open();
        }
    }
}
```

Glavni pogled aplikacije



```
Menu {
    id: contextMenu

    background: Rectangle {
        implicitWidth: 200
        implicitHeight: 40
        color: backgroundColor
    }

    delegate: MenuItem {
        id: menuItem
        contentItem: Text {
            leftPadding: menuItem.indicator.width
            rightPadding: menuItem.arrow.width
            text: menuItem.text
            font: menuItem.font
            opacity: enabled ? 1.0 : 0.3
            color: mainTextColor
            horizontalAlignment: Text.AlignLeft
            verticalAlignment: Text.AlignVCenter
            elide: Text.ElideRight
        }

        background: Rectangle {
            implicitWidth: 200
            implicitHeight: 40
            opacity: enabled ? 1 : 0.3
            color: menuItem.highlighted ? mainAppColor : "transparent"
        }
    }

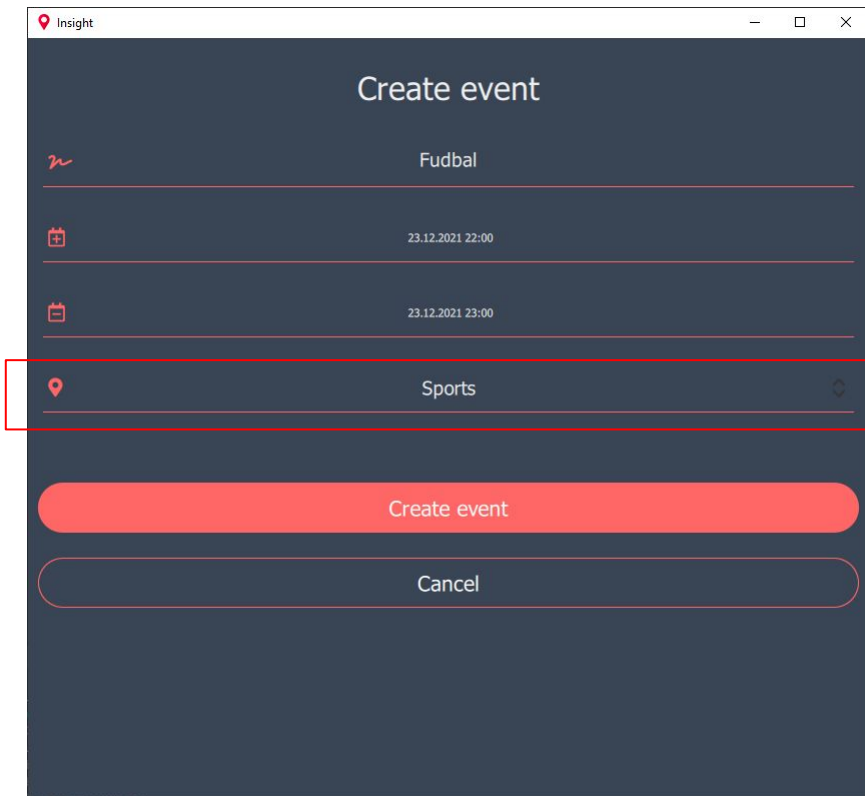
    Action {
        id: insertEvent
        text: 'Insert Event'

        onTriggered: {
            stackView.push("qrc:/EventWrite.qml");
        }
    }

    Action {
        id: logout
        text: 'Logout'

        onTriggered: {
            usermg.logout();
            toLogin();
        }
    }
}
```

Upravljanje događajima



```
ComboBox {
    id: typepicker

    Layout.preferredWidth: parent.width - 20
    Layout.alignment: Qt.AlignHCenter

    background: Rectangle {
        implicitWidth: 200
        implicitHeight: 50
        radius: implicitHeight / 2
        color: "transparent"

        Text {
            text: "\uf3c5"
            font.pointSize: 14
            font.family: "fontawesome"
            color: mainAppColor
            anchors.left: parent.left
            anchors.verticalCenter: parent.verticalCenter
            leftPadding: 10
        }

        Rectangle {
            width: parent.width - 10
            height: 1
            anchors.horizontalCenter: parent.horizontalCenter
            anchors.bottom: parent.bottom
            color: mainAppColor
        }
    }

    contentItem: Text {

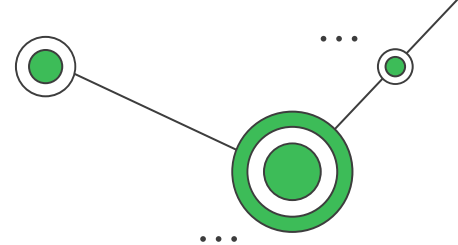
        horizontalAlignment: Text.AlignHCenter
        verticalAlignment: Text.AlignVCenter
        anchors.horizontalCenter: parent.horizontalCenter
        text: typepicker.displayText
        font.pointSize: 14
        font.family: "fontawesome"
        color: mainTextColor
    }

    popup: Popup {
        y: typepicker.height - 1
        width: typepicker.width
        implicitHeight: contentItem.implicitHeight
        padding: 1

        contentItem: ListView {
            clip: true
            implicitHeight: contentHeight
            model: typepicker.popup.visible ? typepicker.delegateModel : null
            currentIndex: typepicker.highlightedIndex
            ScrollIndicator.vertical: ScrollIndicator { }
        }

        background: Rectangle {
            color: mainAppColor
        }
    }
}
```

Upravljanje događajima



Insight

Create event

Fudbal

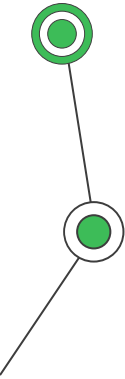
22.12.2021 18:02

2019	October	21	20	58
2020	November	22	21	59
2021	December	23	22	00
2022	January	24	23	01
2023	February	25	24	02

Cancel

Pošto u QML-u ne postoji ništa slično kao `DateTimePicker` kontrola u drugim okvirima, javila se potreba za kreiranjem takve kontrole.

Kontrola je realizovana, po ugledu na slične kontrole koje postoje na Android aplikacijama. Korišćenjem `PopUp`-a i više `Tumble` kontrola koje omogućavaju odabir datuma i vremena.



```

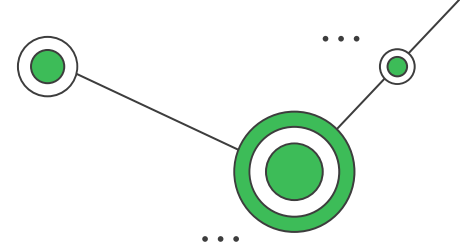
repQuickItem {
    property int eventId: -1
    property string name: ""
    property real lat: 0
    property real lon: 0
    property int type: -1
    property int createdBy: -1
    property date startDate;
    property date endDate;
    //property string imgSource: "qrc:/placeholder.png"

    function getPicture(type) { ... }

    id:placeholderEvent
    anchorPoint.x:image.width/2
    anchorPoint.y:image.height
    sourceItem: Image {
        id: image
        source: getPicture(type)
        height: 24
        width: 24
        MouseArea {
            id: placeholderArea
            anchors.fill: parent
            hoverEnabled: true
            onClicked: {
                var event = [eventId, name, lat, lon, type, createdBy, startDate, endDate];
                setClickedEvent(event);
                console.log(event);
                stackView.push("qrc:/EventInfo.qml");
            }
        }
    }
    coordinate: QtPositioning.coordinate(lat,lon)
}


```


Upravljanje događajima






Insight

Event info

 Fudbal

 22.12.2021 19:04

 22.12.2021 22:04

 Sports 


Update event


Remove event


Cancel



Insight

Event info

 Fudbal

 22.12.2021 19:04

 22.12.2021 22:04

 Sports 

Cancel



Hvala na pažnji!

Pitanja?