АВРОРА

Модуль 4

Тема 4.8

Практика

Создать приложение, позволяющее добавлять и удалять заметки с использованием базы данных (LocalStorage) и отображать их в списке (ListView). Приложение должно отображать на одном экране поле для ввода текста (TextField), кнопку (Button) для добавления заметки в базу данных и непосредственно сам список заметок(ListView.

```
import QtQuick 2.0
import Sailfish.Silica 1.0
import QtQuick.LocalStorage 2.0

Page {
  id: window
  objectName: "mainPage"
  allowedOrientations: Orientation.All

  property string noteText: ""
  property var noteModel: ListModel {
     ListElement { note: "Sample Note 1" }
     ListElement { note: "Sample Note 2" }
  }

  function addNote() {
     if (noteText !== "") {
        noteModel.append({ "note": noteText })
        noteText = ""
     }
  }
```

```
function deleteNote(index) {
    noteModel.remove(index)
}

SilicaFlickable {
    anchors.fill: parent

    Column {
        spacing: Theme.paddingLarge

        TextField {
            id: noteTextField
            placeholderText: "Enter a note"
            onTextChanged: noteText = text
        }

        Button {
            id: addButton
            text: "Add Note"
            onClicked: addNote()
        }

        ListView {
            id: noteListView
            width: parent.width
            height: parent.height / 2
            model: noteModel
            delegate: Item {
```

```
                width: parent.width
                height: Theme.itemSizeMedium
                Row {
                    spacing: Theme.paddingSmall
                    x: 40

                    Label {
                        text: model.note
                        width: parent.width / 2
                        anchors.verticalCenter: parent.verticalCenter
                        color: Theme.primaryColor
                    }

                    Button {
                        id: btnDelete
                        text: "Delete"
                        onClicked: deleteNote(index)
                    }
                }
            }
        }
    }

    function initializeDatabase() {
        var db = LocalStorage.openDatabaseSync("notesDB", "1.0", "Notes Database", 1000000)

        db.transaction(function(tx) {
```

```
        tx.executeSql('CREATE TABLE IF NOT EXISTS notes(note TEXT)')
        console.log("Таблица создана")
    })
 }

 Component.onCompleted: initializeDatabase()
}
```

Основные элементы и функции приложения:

- property string noteText: "": Это свойство хранит текст, введенный пользователем в поле ввода заметки (TextField).
- property var noteModel: ListModel { ... }: Это свойство представляет модель данных для ListView. Оно инициализируется списком ListElement, где каждый элемент содержит заметку.
- function addNote() { ... }: Эта функция вызывается при нажатии на кнопку "Add Note". Она проверяет, что поле ввода заметки не пустое, а затем добавляет новую заметку в модель данных noteModel и очищает поле ввода.
- function deleteNote(index) { ... }: Эта функция вызывается при нажатии на кнопку "Delete" в элементе списка. Она удаляет заметку из модели данных noteModel по указанному индексу.
- SilicaFlickable: Это элемент для создания прокручиваемой области, куда помещается контент страницы.
- TextField: Это поле ввода, где пользователь может вводить текст заметки.
- Button: Это кнопка "Add Note", которая вызывает функцию addNote() при нажатии.

- ListView: Это элемент для отображения списка заметок. Он использует модель данных noteModel и для каждого элемента списка создает делегат типа Item.
- Делегат Item: Это элемент, который отображается для каждого элемента списка. В данном случае, он содержит горизонтальную компоновку Row, в которой есть Label для отображения текста заметки и Button для удаления заметки. При нажатии на кнопку вызывается функция deleteNote() с передачей индекса элемента.
- function initializeDatabase() { ... }: Эта функция вызывается при запуске приложения и инициализирует базу данных LocalStorage. Она открывает или создает базу данных с именем "notesDB" и версией "1.0". Затем выполняет SQL-запрос для создания таблицы "notes" (если она не существует). В данном случае, таблица имеет одно поле "note" типа TEXT, которое будет хранить текст заметки.
- Component.onCompleted: initializeDatabase(): Это свойство вызывает функцию initializeDatabase() при завершении инициализации компонента (страницы).

Таким образом, приложение позволяет пользователю вводить заметки, сохранять их в базе данных и отображать список заметок с возможностью удаления.

АВРОРА

Модуль 4

Тема 4.9

Практика

1. Подключиться с помощью DBus к сервису org.freedesktop.Notifications и получить список возможностей сервера (метод GetCapabilities). Отправить на сервер уведомление при помощи метода Notify. API:

https://specifications.freedesktop.org/notification-spec/notification-spec-latest.html

https://specifications.freedesktop.org/notification-spec/latest/ar01s09.html

2. Зарегистрировать свой сервис, который может вычислять арифметические действия и возвращать результат.

1. Подключиться с помощью DBus к сервису org.freedesktop.Notifications и получить список возможностей сервера (метод GetCapabilities). Отправить на сервер уведомление при помощи метода Notify. API:

.pro

```
QT += dbus
```

.spec

```
Requires: nemo-qml-plugin-dbus-qt5
```

```
import QtQuick 2.0
import Sailfish.Silica 1.0
import Nemo.DBus 2.0
import Nemo.Notifications 1.0

Page{

  Item {
    DBusInterface {
        id: profiled

        service: 'org.freedesktop.Notifications'
        iface: 'org.freedesktop.Notifications'
        path: '/org/freedesktop/Notifications'
    }

    Component.onCompleted: {
        // Вызывается метод '"GetCapabilities"' без аргументов и,
        // когда он вернёт значение, следует вызвать переданную функцию обратного вызова
        profiled.call('GetCapabilities', [], function (result) {
            // Этот код будет исполнен после получения результата вызванного метода
            console.log('Got Capabilities: ' + result);
```

```
        });
    }
  }

  Button {

    Notification {
        id: notification1
        appName: "Example App"
        summary: "Notification Summary"
        body: "Notification Body"
        icon: "icon-s-do-it"
        remoteActions: [
          {
            "app_name": "default",
            "replaces_id": "Do something",
            "app_icon": "icon-s-do-it",
            "summary": "org.freedesktop.Notifications",
            "body": "/example",
            "actions": "org.freedesktop.Notifications",
            "hints": "doSomething",
            "expire_timeout": [ "argument", 1 ]
          }
        ]
        onClosed: {
          if (reason === Notification.Expired) {
            console.log("Notification expired");
          } else if (reason === Notification.DismissedByUser) {
            console.log("Notification dismissed by user");
          }
        }
      }
    text: "Application notification" + (notification1.replacesId ? " ID:" +
notification1.replacesId : "")
    onClicked: notification1.publish()
  }
}
```

```
[D] :59 - Got Capabilities: body,actions,persistence,sound,x-nemo-item-count,x-nemo-
timestamp,x-nemo-preview-body,x-nemo-preview-summary,x-nemo-remote-actions,x-nemo-user-
removable,x-nemo-get-notifications
[W] unknown:0 - Invalid remote action specification: QVariant(QVariantMap, QMap(("actions",
QVariant(QString, "org.freedesktop.Notifications"))("app_icon", QVariant(QString, "icon-s-do-
it"))("app_name", QVariant(QString, "default"))("body", QVariant(QString, "/example"))
("expire_timeout", QVariant(QVariantList, (QVariant(QString, "argument"), QVariant(int, 1))))
("hints", QVariant(QString, "doSomething"))("replaces_id", QVariant(QString, "Do something"))
("summary", QVariant(QString, "org.freedesktop.Notifications"))))
```

remoteActions : list<variant>
Это список зарегистрированных дистанционных действий, которые могут быть выполнены при получении уведомления.

Удалённые действия указываются в виде списка объектов с обязательными свойствами 'name', 'displayName, 'icon', 'service', 'path', 'iface', 'method' и 'arguments'. Требуется всегда указывать 'Name', а также 'displayName', если действиет отличается от "default" или "app".

Описание кода:
Блок с Item. Определяет элемент Item, внутри которого находится компонент DBusInterface. DBusInterface позволяет взаимодействовать с сервисом посредством DBus. Здесь происходит вызов метода GetCapabilities сервиса org.freedesktop.Notifications без аргументов. После получения результата вызова метода будет выполнена функция обратного вызова, которая выводит результат в консоль.

Далее идет элемент Button с вложенным компонентом Notification. Когда кнопка нажимается, вызывается метод publish() у компонента Notification, чтобы опубликовать уведомление.

Компонент Notification определяет параметры уведомления, такие как appName, summary, body и icon. В данном случае уведомление содержит информацию о примере приложения, его кратком описании и теле уведомления, а также иконку, указанную как "icon-s-do-it". Уведомление также содержит список удаленных действий (remoteActions), который определяет дополнительные параметры для действий, связанных с уведомлением.

Блок onClosed в Notification определяет обработчик события закрытия уведомления. В данном случае, если уведомление было закрыто по истечении времени (Notification.Expired), будет выведено сообщение в консоль. Если уведомление было закрыто пользователем (Notification.DismissedByUser), также будет выведено сообщение.

Далее задается текст для кнопки, который включает идентификатор уведомления (replacesId), если он присутствует.

2. Зарегистрировать свой сервис, который может вычислять арифметические действия и возвращать результат.

```
import QtQuick 2.0
import Sailfish.Silica 1.0
import Nemo.DBus 2.0
import Nemo.Notifications 1.0

Page{
  Item {
     DBusAdaptor {
        id: dbuscalculator

        property bool needUpdate: true

        service: 'org.freedesktop.dbuscalculator'
        iface: 'org.freedesktop.dbuscalculator'
        path: '/org/freedesktop/dbuscalculator'

        xml: '  <interface name="org.freedesktop.dbuscalculator">\n' +
            '    <method name="add" />\n' +
            '    <method name="sub" />\n' +
            '  </interface>\n'

        function sum(a, b) {
```

```
            console.log("sum")
            return a+b
        }

        function sub(a, b) {
            console.log("sub")
            return a-b
        }

    }
}



Item {
  DBusInterface {
      id: dbuscalculatorinterface

      service: 'org.freedesktop.dbuscalculator'
      iface: 'org.freedesktop.dbuscalculator'
      path: '/org/freedesktop/dbuscalculator'

      signalsEnabled: true

      function sum() {
          dbuscalculatorinterface.call('sum', [3, 2], function (result) {
                  console.log('Answer: ' + result);
          });
      }

      function sub() {
          dbuscalculatorinterface.call('sub', [3, 2], function (result) {
                  console.log('Answer: ' + result);
          });
      }
  }
```

```
  }

  Button {
    anchors.centerIn: parent
    text: "Press me"
    onClicked: {
      dbuscalculatorinterface.sum()
      dbuscalculatorinterface.sub()
    }
  }
}
```

Описание кода:

Создаем элемент Item с вложенным компонентом DBusAdaptor. DBusAdaptor позволяет создавать сервисы DBus. Здесь создается сервис org.freedesktop.dbuscalculator с интерфейсом org.freedesktop.dbuscalculator и путем /org/freedesktop/dbuscalculator.

Затем в блоке xml определена структура интерфейса DBus, включающая два метода: add и sub.

Функции sum и sub определены для обработки вызовов методов add и sub соответственно. Когда эти методы вызываются извне, они выводят сообщения в консоль и возвращают сумму или разность аргументов.

Создаем еще один Item с DBusInterface. Элемент DBusInterface позволяет взаимодействовать с сервисом DBus org.freedesktop.dbuscalculator. Он также определяет две функции: sum и sub. Когда эти функции вызываются, они отправляют соответствующие запросы к методам sum и sub сервиса, используя метод call. После получения ответа, результат выводится в консоль.

Затем создаем элемент Button, который располагается по центру родительского элемента и имеет надпись "Press me". При нажатии на кнопку вызываются функции sum и sub объекта dbuscalculatorinterface, что приводит к отправке соответствующих запросов DBus сервису.

Модуль 4. Использование системных API

Тема 4.10. QML-плагины ОС Аврора

Глоссарий

QML-плагин KeepAlive предоставляет типы для предотвращения гашения дисплея, предотвращения приостановки работы системы и планирования пробуждения из приостановленного состояния.

QML-тип DisplayBlanking

Отслеживает состояние MCE по D-Bus и предоставляет текущее состояние экрана как свойство QML.

QML-тип BackgroundJob

Предоставляет простую абстракцию для механизмов D-Bus, которые необходимы для предотвращения гашения экрана (если это разрешено политиками более низкого уровня).

Предоставляет абстракцию для планирования задач, которые могут вывести систему из приостановленного состояния и предотвратить приостановку системы во время обработки пробуждения.

QML-тип KeepAlive

Предоставляет простую абстракцию для механизмов D-Bus, которые необходимы для предотвращения приостановки работы устройства (если это разрешено политиками более низкого уровня).

Thumbnailer - это программный инструмент, который позволяет создавать и отображать миниатюрные изображения в реальном времени. Он используется для создания уменьшенных изображений, которые могут быть использованы в качестве предварительного просмотра больших изображений, а также в различных графических приложениях, таких как галереи изображений, редакторы фотографий, веб-сайты и другие приложения.

Модуль Thumbnailer API предоставляет для выборки эскизов изображений и видео в приложениях. Эскизы генерируются по требованию и кэшируются на диске для быстрого доступа.

QML-тип Thumbnail

Вместо элементов типа Qt Quick Image для отображения эскизов изображений и видео может использоваться специализированный QML-тип Thumbnail. Плагин Thumbnailer предоставляет дополнительный API, с помощью которого можно задавать приоритет запросам, а также получать текущее состояние процесса генерации эскиза. Загруженные эскизы хранятся в локальном дисковом кэше, что ускоряет их последующую загрузку, особенно если исходное изображение или видео достаточно большого размера.

QML-плагин Notifications

Данный плагин предоставляет C++/QML-классы для публикации уведомлений в среде Nemo.

QML-тип Notification

Тип Notification представляет собой удобный способ работы с уведомлениями. Тип основан на спецификации уведомлений рабочего стола (Desktop Notifications Specification), реализованной в Nemo.

Этот тип позволяет клиентским приложениям создавать уведомления, которые могут использоваться для обмена данными с диспетчером уведомлений домашнего экрана через D-Bus. Это упрощает процесс создания, отображения и закрытия уведомлений, так как сам тип управляет всеми необходимыми процедурами обмена данными.

Тип Notification представляет собой удобный способ работы с уведомлениями. Тип основан на спецификации уведомлений рабочего стола (Desktop Notifications Specification), реализованной в Nemo.

Этот класс позволяет клиентским приложениям создавать уведомления, которые могут использоваться для обмена данными с диспетчером уведомлений домашнего экрана через D-Bus. Это упрощает процесс создания, отображения и закрытия уведомлений, так как сам класс управляет всеми необходимыми процедурами обмена данными.

QML-модуль Share позволяет загружать пользовательский интерфейс для обмена данными и отображать список средств отправки. С помощью этого модуля пользователь имеет возможность делиться файлами и прочим контентом, отправляя данные любым доступным способом, например, по Bluetooth, SMS, электронной почте и т. д.

Пересылаемые файлы могут иметь любой MIME-тип, который определяет список возможных средств для обмена данными.

QML-типы Share:

- ShareAction
- ShareProvider
- ShareResource

QML-тип ShareAction

QML-тип ShareProvider

Принимает отправленные файлы и с помощью ShareProvider::triggered оповещает о передаче данных другим приложением.

QML-тип ShareResource

Тип для ресурсов из ShareProvider::triggered. Данный тип нельзя создать напрямую из QML.