МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«Нижегородский государственный университет им. Н.И. Лобачевского»
Национальный исследовательский университет

Институт информационных технологий, математики и механики
Кафедра программной инженерии

ОТЧЕТ

по дисциплине «Разработка мобильных приложений»

«Управление данными и использование шаблона MVC»

Выполнил:
студент группы 381706-1
Денисов В. Л.

Проверил:
доцент кафедры программной инженерии
Борисов Н. А.

Нижний Новгород
2020.

Содержание

1 Цели

Целью данной лабораторной работы является изучение использования различных моделей для отображения данных в прокручиваемых списках, взаимодействия с базой данных и управления настройками приложения.

2 Постановка задачи

1. Создать приложение, которое позволяет отображать список из прямоугольников с использованием ListModel. В модели должны настраиваться цвет фона и текста внутри прямоугольника. Текст содержит название цвета фона прямоугольника.
2. Создать приложение, которое позволяет отображать список из прямоугольников. Нажатие на кнопку над списком добавит новый элемент. Нажатие на элемент в списке удалит его из списка. В прямоугольниках должен отображаться порядковый номер, присваиваемый при добавлении в список. При удалении элементов порядковые номера у добавленных прямоугольников остаются неизменными.
3. Выполнить задание из пункта 1 путём использования javascript-модели.
4. Получить и отобразить курсы валют из ресурса ЦБ РФ по адресу http://www.cbr.ru/scripts/XML_daily.asp.
5. Выполнить задание из пункта 4 путём использования XMLHttpRequest.
6. Создать приложение, позволяющее добавлять и удалять заметки с использованием базы данных и отображать их в списке. Текстовое поле служит для ввода текста, кнопка для добавления заметки, нажатие на заметку удалит её.
7. Создать приложение с текстовым полем и полем с флажком, значение которых сохраняется в настройках приложения с помощью ConfigurationValue.
8. Выполнить задание из пункта 7 путём использования ConfigurationGroup.

3 Решение поставленных задач

Создадим проект со стандартной заготовкой приложения, где файлом главной страницы приложения будет являться *FirstPage.qml*, для каждого отдельного задания из постановки задачи тоже будет свой файл.

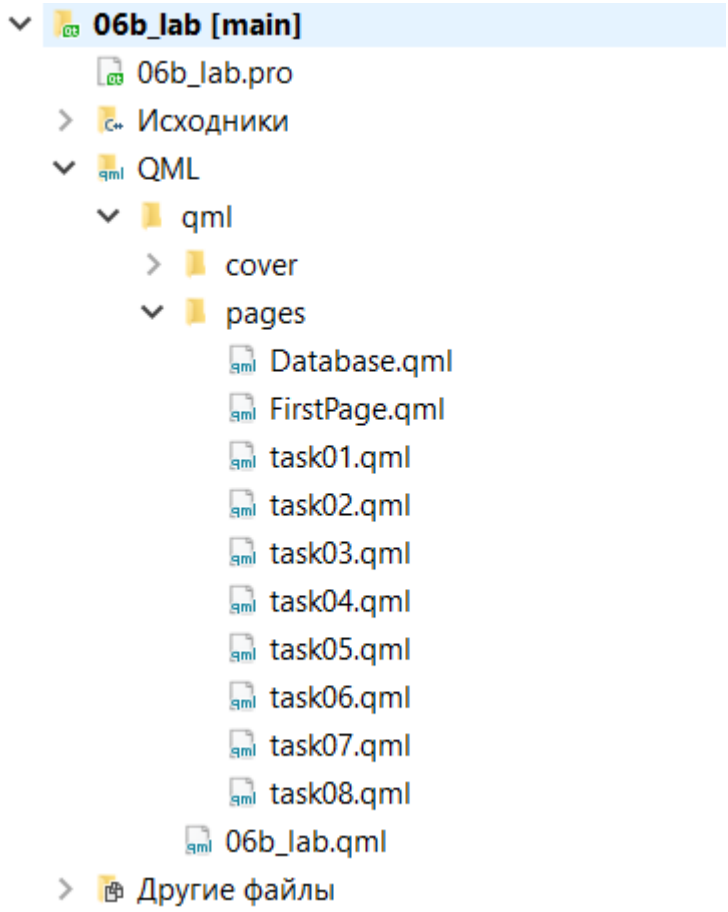

Рисунок 1 Структура проекта.

Вызов демонстрации каждого задания будем производить путем нажатия соответствующей кнопки, расположенной на *FirstPage*. Страница с выбранным заданием будет отправлена в *pageStack*.

1. Для отображения списка прямоугольников используем элемент *SilicaListView*, в котором в качестве атрибута *model* устанавливаем *ListModel*, включающий в себя *ListElement*'ы, в которых хранятся требуемые параметры: *bgcolor* – цвет фона, *textcolor* – цвет текста, *text* – текст. Атрибуту *delegate* у *SilicaListView* назначаем *Rectangle* с *Label* внутри, которые будут использовать соответствующие параметры модели.

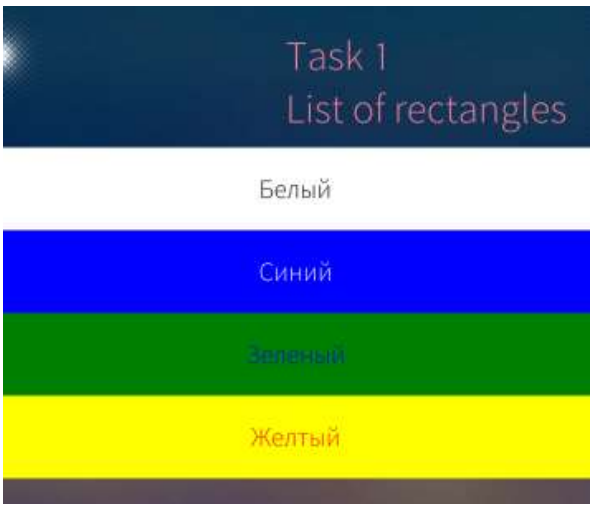

Рисунок 2 Список прямоугольников с использованием ListModel

2. Решение поставленной задачи осуществляем при помощи *SilicaListView* – будет отображать список, где в атрибутах: *model* – хранятся элементы списка в контейнере *ListModel*, *delegate* – структура их отображения в виде Rectangle. Для добавления элементов с список вводим свойство *property int count*, которое будет только увеличиваться при добавлении новых элементов в список и не уменьшаться при их удалении – таким образом сохраним требуемую индексацию. Добавление осуществляем путем вызова *dataModel.append({ text: "Item #" + column.count++ })* – присоединяем новый элемент к *ListModel*. Удаление по событию *onClicked* при нажатии в области *MouseArea*, замещающей весь *Rectangle*, который представляет собой отображение элемента списка – выполняем метод *dataModel.remove(model.index, 1)*, где *model.index* – индекс элемента в списке, по которому было произведено нажатие, 1 – количество удаляемых элементов.

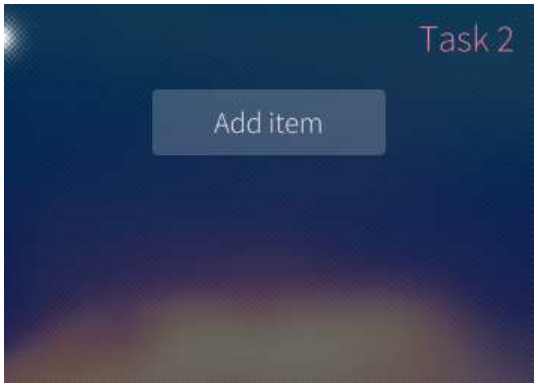

Рисунок 3 Список прямоугольников (пустой)

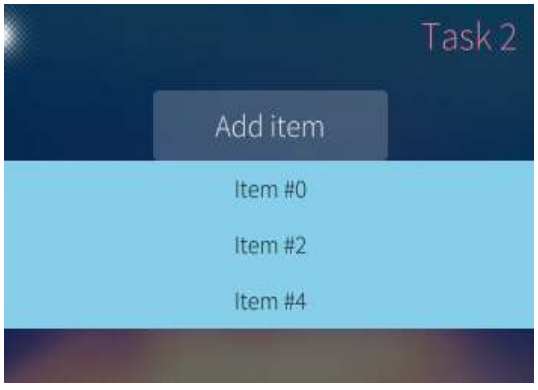

Рисунок 4 Список прямоугольников (после удаления элементов и последующего добавления новых элементов)

3. Подход к отображению элементов списка прямоугольников аналогичен пункту 1. Однако вместо *ListModel* вводим переменную *property var modelData*, которая представляет собой массив, элементами которого являются словари, хранящие требуемые параметры в виде *“ключ”:”значение”*. В качестве ключей используются: *bgcolor* – цвет фона, *textcolor* – цвет текста, *text* – текст.

Рисунок 5 Список прямоугольников с использованием javascript-модели

4. Курсы валют в приложении представим в виде списка – для этого используем *SilicaListView*. В качестве *model* устанавливаем объект *XmlListModel*, которому назначим ряд атрибутов. Перечислим их: *source* – адрес расположения *xml*-файла, из которого будем получать информацию о курсах валют, *query* – к каким конкретно элементам требуется обращаться (в нашем случае идем по всем валютам *“/ValCurs/Valute”*), *XmlRole* – что конкретно будет доступно нашей модели и как это интерпретировать (нам потребуется название валюты *"Name"* и ее курс *“Value”*).

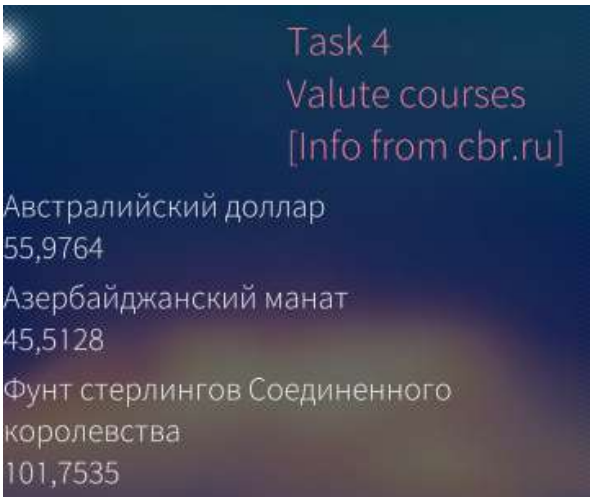

Рисунок 6 Курс валют ЦБ РФ с использованием XmlListModel

5. Решение этого пункта во многом похоже на пункт 4. Однако в этом случае не будем устанавливать атрибут *source*. Для получения исходных данных для дальнейшей обработки создадим функцию *loadCourses()*, вызов которой осуществляется по событию *Component.onCompleted* на странице приложения. Функция *loadCourses()* создает объект *XMLHttpRequest*, с помощью которого выполняется GET-запрос по тому же адресу расположения *xml*-файла. Если запрос был успешно обработан на стороне сервера, то полученный ответ, представляющий собой *XML*-структуру, преобразуется в текст из исходной кодировки Windows-1251 в кодировку UTF-8 для корректного отображения в приложении. Выполняется присваивание ответа в свойство *xmlModel.xml*, хранящее данные *XML*-модели.

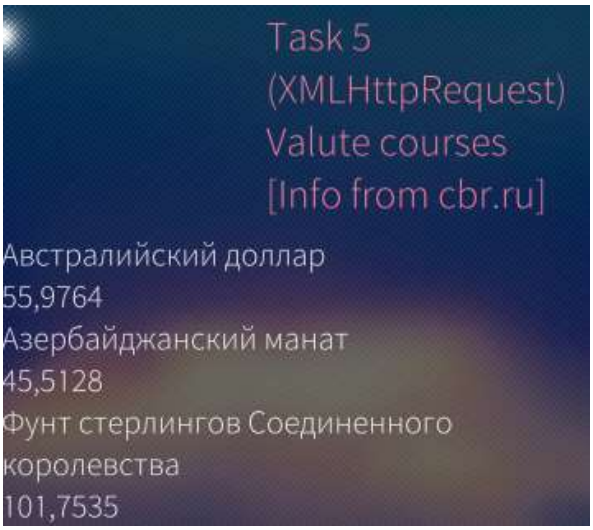

Рисунок 7 Курс валют ЦБ РФ с использованием XMLHttpRequest

6. Для выполнения этого задания отдельно вынесем весь необходимый функционал для работы с базой данных – разместим его в файле *Database.qml*. Используем контейнер *Item*, внутри которого создаем свойство *property var db*, которое будет представлять базу данных. В обработчике *Component.onCompleted* выполняем подключение к базе данных *“tasks”* путём вызова метода *LocalStorage.openDatabaseSync("tasks", "1.0")*. Исходный код функций для создания таблицы заметок в базе данных, добавления и удаления заметок представлен в приложении. Все эти функции представляют собой передачу соответствующего SQL-запроса внутрь транзакции, инициируемой путем *db.transaction* для корректной работы с базой данных.

   Затем непосредственно на странице приложения, представляющей функционал для работы с заметками, создаем ряд элементов: *Database* – контейнер базы данных, описанный в одноименном файле *Database.qml*; *ListModel* – список заметок, *SilicaListView* – отображение заметок; *TextField* – поле для ввода текста новой заметки;

Button – кнопка для добавления новой заметки. Для работы с базой обращаемся по *id* к контейнеру, который ее представляет, и вызываем требуемый метод.

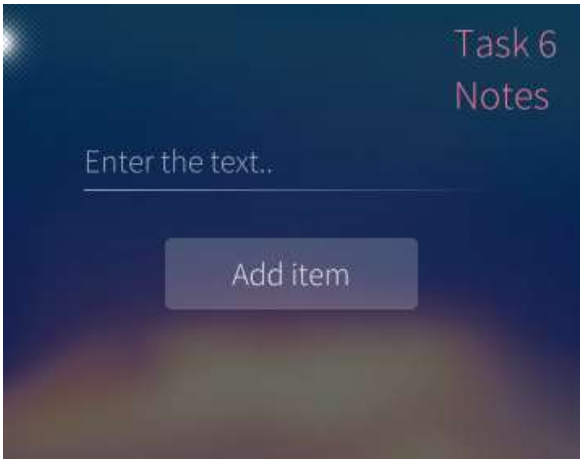

Рисунок 8 Пустой список заметок

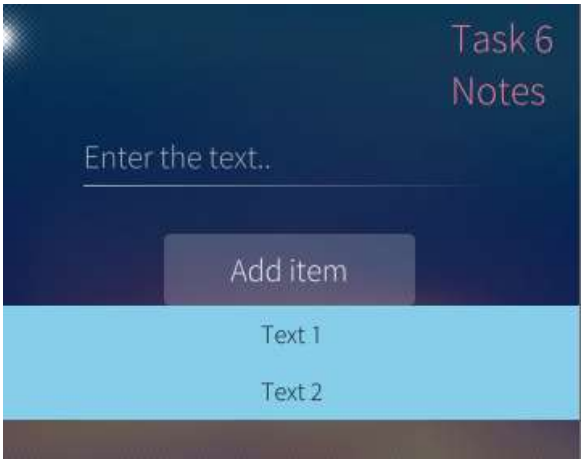

Рисунок 9 Несколько заметок добавлено в базу данных

```
[D] :44 - Selected from database...
[D] insertTask:24 - Insert task with 'name':   Text 1
[D] :44 - Selected from database...
[D] :47 - 2  Text 1
[D] insertTask:24 - Insert task with 'name':   Text 2
[D] :44 - Selected from database...
[D] :47 - 2  Text 1
[D] :47 - 3  Text 2
```

Рисунок 10 Лог в консоли при работе с базой данных

7. Для сохранения требуемых значений при помощи *ConfigurationValue* потребуется указать атрибут *key* – место, где будет храниться соответствующий параметр. При нажатии на кнопку для сохранения настроек, обращаемся по *id* к нужному *ConfigurationValue*, обновляем значение атрибута *value* – текущее значение, хранимое по пути *key*, на то, которое получаем из *TextField*'а или *TextSwitch*'а.

Рисунок 11 Сохранение настроек при помощи ConfigurationValue

8. Принцип для обновления параметров аналогичен пункту 7. Отличие непосредственно в хранении параметров. Для использования *ConfigurationGroup* необходимо установить атрибут *path* – аналогичен *key* из пункта 7, а также добавить столько свойств, сколько параметров потребуется хранить (в нашем случае 2: *property string text* и *property bool flag*).

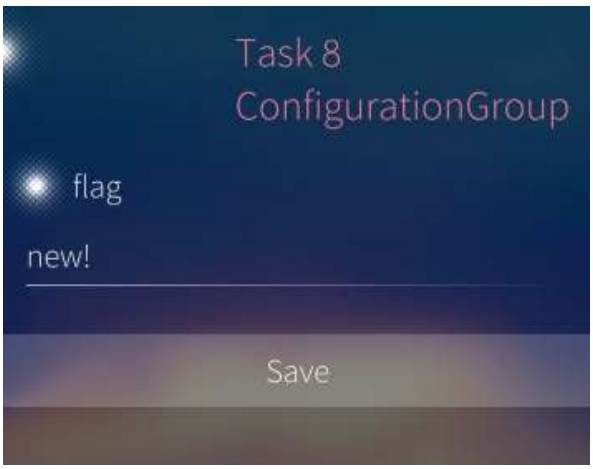

Рисунок 12 Сохранение настроек при помощи ConfigurationGroup

4 Приложение

4.1 Файл FirstPage.qml

```
import QtQuick 2.0
import Sailfish.Silica 1.0

Page {
    id: page
    allowedOrientations: Orientation.All

    SilicaFlickable {
        anchors.fill: parent
        contentHeight: column.height

        Column {
            id: column
            width: page.width
            spacing: Theme.paddingLarge
            PageHeader {
                title: qsTr("Main Page")
            }
            Button {
                text: qsTr("Task 1")
                onClicked: pageStack.push(Qt.resolvedUrl("task01.qml"))
                anchors.horizontalCenter: parent.horizontalCenter
            }
            Button {
                text: qsTr("Task 2")
                onClicked: pageStack.push(Qt.resolvedUrl("task02.qml"))
                anchors.horizontalCenter: parent.horizontalCenter
            }
            Button {
                text: qsTr("Task 3")
                onClicked: pageStack.push(Qt.resolvedUrl("task03.qml"))
                anchors.horizontalCenter: parent.horizontalCenter
            }
            Button {
                text: qsTr("Task 4")
                onClicked: pageStack.push(Qt.resolvedUrl("task04.qml"))
                anchors.horizontalCenter: parent.horizontalCenter
            }
            Button {
                text: qsTr("Task 5")
                onClicked: pageStack.push(Qt.resolvedUrl("task05.qml"))
                anchors.horizontalCenter: parent.horizontalCenter
            }
            Button {
                text: qsTr("Task 6")
                onClicked: pageStack.push(Qt.resolvedUrl("task06.qml"))
                anchors.horizontalCenter: parent.horizontalCenter
            }
            Button {
                text: qsTr("Task 7")
                onClicked: pageStack.push(Qt.resolvedUrl("task07.qml"))
                anchors.horizontalCenter: parent.horizontalCenter
            }
            Button {
                text: qsTr("Task 8")
                onClicked: pageStack.push(Qt.resolvedUrl("task08.qml"))
                anchors.horizontalCenter: parent.horizontalCenter
            }
```

```
        } // Column
    } // SilicaFlickable
} // Page
```

4.2 Файл task01.qml

```
import QtQuick 2.0
import Sailfish.Silica 1.0

Page {
    ListModel {
        id: dataModel
            ListElement { bgcolor: "white"; textcolor:"black"; text: "Белый"
}
            ListElement { bgcolor: "blue"; textcolor:"white"; text: "Синий" }
            ListElement { bgcolor: "green";  textcolor:"blue";  text:
"Зеленый" }
            ListElement { bgcolor: "yellow"; textcolor:"red"; text: "Желтый"
}
    } // ListModel

    SilicaListView {
        anchors.fill: parent
        header: PageHeader {
            title: "Task 1" + "\n" + "List of rectangles"
        }
        model: dataModel
        delegate: Rectangle {
            width: parent.width
            height: 100
            color: model.bgcolor
            Text {
                anchors.centerIn: parent
                text: model.text
                color: model.textcolor
            }
        }
    } // SilicaListView
} // Page
```

4.3 Файл task02.qml

```
import QtQuick 2.0
import Sailfish.Silica 1.0

Page {
    id: page;

    Column {
        id:column;
        anchors.fill: parent;
        PageHeader {
            title: qsTr("Task 2")
        }
        property int count: 0

        Button {
            id: addButton
            anchors.horizontalCenter: parent.horizontalCenter
            anchors.bottom: list.Top
            text: "Add item"
            onClicked: {
```

```
                dataModel.append({ text: "Item #" + column.count++ });
            }
        } // Button

        ListModel {
            id: dataModel
        }

        SilicaListView {
            id:list
            width: parent.width;
            height: parent.height - addButton.height;
            model: dataModel
            delegate: Rectangle {
                width: parent.width; height: 70; color: "skyblue";
                Text {
                    anchors.centerIn: parent
                    text: model.text
                }
                MouseArea {
                    id:mouse
                    anchors.fill: parent
                    onClicked: {
                        dataModel.remove(model.index, 1);
                    }
                }
            } // delegate
        } // SilicaListView
    } // Column
} // Page
```

4.4 Файл task03.qml

```
import QtQuick 2.0
import Sailfish.Silica 1.0

Page {
    property var modelData: [
        { bgcolor: "white", textcolor: "black", text: "Белый" },
        { bgcolor: "blue", textcolor: "white", text: "Синий" },
        { bgcolor: "green"},
        { bgcolor: "yellow", textcolor: "red", text: "Желтый" }
    ]

    SilicaListView {
        id: listView
        anchors.fill: parent
        header: PageHeader {
            title: qsTr("Task 3" + "\n" + "List of rectangles (JS)")
        }
        model: modelData
        delegate: Rectangle {
            id: rec
            width: parent.width
            height: 100
            color: modelData.bgcolor
            Text {
                anchors.centerIn: parent
                text: modelData.text || "Текст не указан"
                color: modelData.textcolor || "black"
            }
        } // delegate
    } // SilicaListView
```

```
} // Page
```

4.5 Файл task04.qml

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

Page {
    id: page

    XmlListModel {
        id: xmlmodel
        source: "http://www.cbr.ru/scripts/XML_daily.asp"
        query: "/ValCurs/Valute"
        XmlRole { name: "Name"; query: "Name/string()"; }
        XmlRole { name: "Value"; query: "Value/string()" }
    }

    SilicaListView {
        id: slv
        model: xmlmodel
        anchors.fill: parent
        header: PageHeader {
            title: qsTr("Task 4" + "\n" +
                        "Valute courses" + "\n" +
                        "[Info from cbr.ru]")
        }
        spacing: 10
        delegate: Column {
            width: parent.width
            Label {
                width: parent.width;
                text: Name + "\n" + Value
                wrapMode: Text.WrapAtWordBoundaryOrAnywhere
            }
        }
    } // SilicaListView
} // Page
```

4.6 Файл task05.qml

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

Page {
    id: page

    Component.onCompleted: {
        loadCourses()
    }

    function loadCourses() {
        var xhr = new XMLHttpRequest();
        xhr.open('GET', "http://www.cbr.ru/scripts/XML_daily.asp", true);

        xhr.onreadystatechange = function() {
            if (xhr.readyState === XMLHttpRequest.DONE) {
                xmlModel.xml = xhr.responseText.replace("windows-1251","utf-
8");
            }
        }
```

```
        xhr.send();
    }

    XmlListModel {
        id: xmlModel
        query: "/ValCurs/Valute"
        XmlRole{ name: "Name"; query: "Name/string()";}
        XmlRole{ name:"Value"; query:"Value/string()";}
    }

    SilicaListView {
        anchors.fill: parent
        header: PageHeader {
            title: qsTr("Task 5" + "\n" +
                        "(XMLHttpRequest)" + "\n" +
                        "Valute courses" + "\n" +
                        "[Info from cbr.ru]")
        }
        model: xmlModel
        delegate: Column {
            width: parent.width
            Label {
                width: parent.width;
                text: Name + "\n" + Value
                wrapMode: Text.WrapAtWordBoundaryOrAnywhere
            }
        }
    } // SilicaListView
} // Page
```

4.7 Файл task06.qml

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

Page {
    id: page;

    Database { id: db }

    Component.onCompleted: db.select();

    Column {
        id:column;
        anchors.fill: parent;
        PageHeader {
           title: qsTr("Task 6" + "\n" +"Notes")
        }

        TextField {
            id: txtfield
            anchors.horizontalCenter: parent.horizontalCenter
            width: parent.width / 5 * 4
            placeholderText: "Enter the text.."
        }

        Button {
            id: addButton
            anchors.horizontalCenter: parent.horizontalCenter
            anchors.bottom: list.Top
            text: "Add item"
            onClicked:{
```

```
                if (txtfield.text !== '') {
                    db.insertTask(txtfield.text);
                    db.select();
                }
            }
        }

        ListModel {
            id: tasksListModel
        }

        SilicaListView {
            id:list
            width: parent.width;
            height: parent.height - addButton.height;
            model: tasksListModel
            delegate: Rectangle {
                width: parent.width; height: 70; color: "skyblue";
                Text {
                    anchors.centerIn: parent
                    text: name
                }
                MouseArea {
                    id:mouse
                    anchors.fill: parent
                    onClicked: {
                        db.deleteTask(id);
                        db.select();
                    }
                }
            } // delegate
        } // SilicaListView
    } // Column
} // Page
```

4.7.1 Файл Database.qml

```
import QtQuick 2.0
import QtQuick.LocalStorage 2.0

Item {
    property var db;

    Component.onCompleted: {
        db = LocalStorage.openDatabaseSync("tasks", "1.0");
        createTable();
    }

    function createTable() {
        db.transaction(function(tx) {
            tx.executeSql("CREATE TABLE IF NOT EXISTS tasks (
                id INTEGER PRIMARY KEY AUTOINCREMENT,"
                + "name TEXT NOT NULL);");
        });
    } // createTable()

    function insertTask(name) {
        db.transaction(function(tx) {
            tx.executeSql("INSERT INTO tasks (name) VALUES(?);", [name]);
        });
        console.log("Insert task with 'name': ", name)
    } // insertTask()

    function deleteTask(id) {
```

```
        db.transaction(function(tx) {
            tx.executeSql("DELETE FROM tasks WHERE id = ?;", [id]);
        });
        console.log("Deleted task with 'id': ", id)
    } // deleteTask()

    function retrieveTasks(callback) {
        db.transaction(function(tx) {
            var result = tx.executeSql("SELECT * FROM tasks;");
            callback(result.rows);
        });
    } // retrieveTasks()

    function select() {
        tasksListModel.clear();
        retrieveTasks(function(tasks) {
            console.log("Selected from database...")
             for (var i = 0; i < tasks.length; i++) {
                var tsk = tasks.item(i);
                console.log(tsk.id + " " + tsk.name)
                tasksListModel.append({id: tsk.id, name: tsk.name});
             }
         });
     } // select()
} // Item
```

4.8 Файл task07.qml

```
import QtQuick 2.0
import Sailfish.Silica 1.0
import Nemo.Configuration 1.0
Page {
    id: page

    ConfigurationValue {
        id: set_for_text
        key: "/apps/06b_lab/setting_for_text"
        defaultValue: "none"
    }
    ConfigurationValue{
        id: set_for_switch
        key: "/apps/06b_lab/setting_for_switch"
        defaultValue: false
    }

    Column {
        width: parent.width
        PageHeader {
            title: qsTr("Task 7" + "\n" +"ConfigurationValues")
        }

        TextSwitch {
            id: textswitch
            text: "flag"
            checked: set_for_switch.value
        }

        TextField {
            id:textfield
            width: parent.width
            text: set_for_text.value
        }
```

```
        Button {
             width: parent.width
             text: "Save configs"
             onClicked: {
                 set_for_text.value = textfield.text
                 set_for_switch.value = textswitch.checked
             }
         }
    } // Column
}
```

4.9 Файл task08.qml

```
import QtQuick 2.0
import Sailfish.Silica 1.0
import Nemo.Configuration 1.0
Page {
    id: page

    ConfigurationGroup {
        id: settings
        path: "/apps/06b_lab/settings"
        property string text: "none"
        property bool flag: false
    }

    Column {
        width: parent.width
        PageHeader {
            title: qsTr("Task 8" + "\n" + "ConfigurationGroup")
        }

        TextSwitch {
            id: textswitch
            text: "flag"
            checked: settings.flag
        }

        TextField {
            id: textfield
            width: parent.width
            text: settings.text
        }

        Button {
             width: parent.width
             text: "Save"
             onClicked: {
                settings.text = textfield.text
                settings.flag = textswitch.checked
             }
         }
    } // Column
}
```

5 Используемая литература

1. Документация QT – https://doc.qt.io/qt-5/qmake-project-files.html