Комитет по образованию Правительства Санкт-Петербурга

**САНКТ-ПЕТЕРБРУГСКИЙ КОЛЛЕДЖ ИНФОРМАЦИОННЫХ   
ТЕХНОЛОГИЙ**

**Отчет по практической работе  
МДК 01.02 «Разработка мобильных приложений»  
Разработка интерактивного графического приложения**

Выполнил

студент группы 493:

Лукьянов И. А.

Преподаватель: Фомин А.В.

Санкт-Петербург 2022

Структура базы данных

База данных состоит из 3 таблиц:

1. Graph – хранит данные о графах.
2. Node – хранит данные об узлах графа.
3. Link – хранит данные о связях графа.

ER диаграмма представлена на рисунке 1.

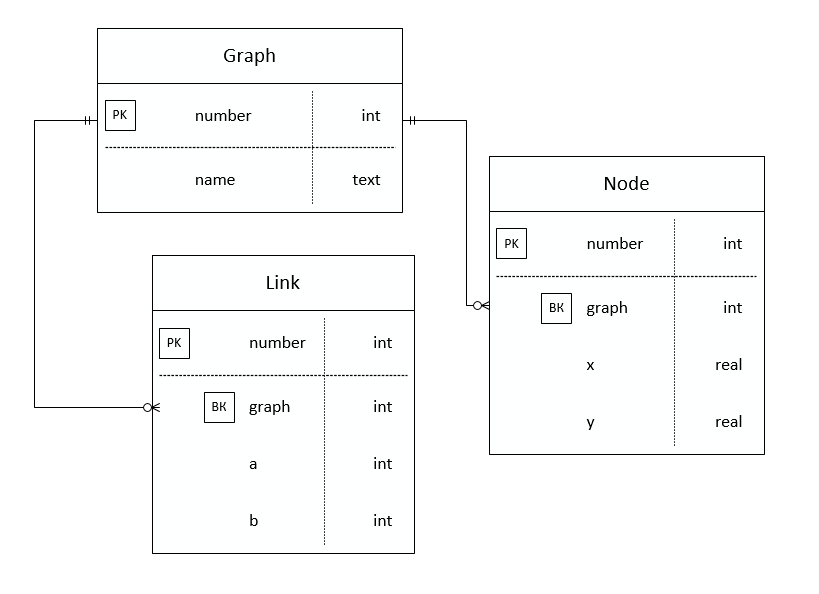


Рисунок 1 – ER диаграмма базы данных

**Таблица Graph**

Содержит сведения о графах приложения. Таблица состоит из двух столбцов:

1. number –номер графа.
2. name – название графа.

Подробное описание столбцов представлено на рисунке 2.

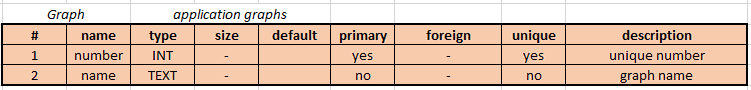


Рисунок 2 – Описание столбцов таблицы Graph

**Таблица Node**

Содержит сведения об узлах графа. Таблица состоит из четырех столбцов:

1. number – номер узла.
2. graph – номер графа, на котором находится узел.
3. x – координата по горизонтали.
4. y – координата по вертикали.

Подробное описание столбцов представлено на рисунке 3.

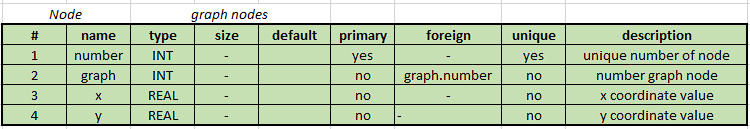


Рисунок 3 – Описание столбцов таблицы Node

**Таблица Link**

Содержит сведения о связях графа. Таблица состоит из четырех столбцов:

1. number – номер связи.
2. graph – номер графа, на котором находится связь.
3. a – номер узла, от которого происходит связь.
4. b – номер узла, к которому происходит связь.

Подробное описание столбцов представлено на рисунке 4.

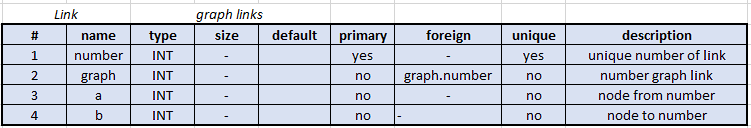


Рисунок 4 – Описание столбцов таблицы Link

Интерфейс приложения

Приложение состоит из 2 форм:

1. Main Form: стартовая форма, служит для управления узлами и связями, а также переходу к форме графов.
2. Graph Form: форма, на которой находятся сохраненные графы с возможностями сохранения, загрузки, переименования, копирования и удаления графа.
3. Node: форма для задания свойств узла, а именно имени и координат.
4. Link: форма для задания значения связи.

**Форма Main Menu**

На рисунке 5 показан макет внешнего вида главной формы.

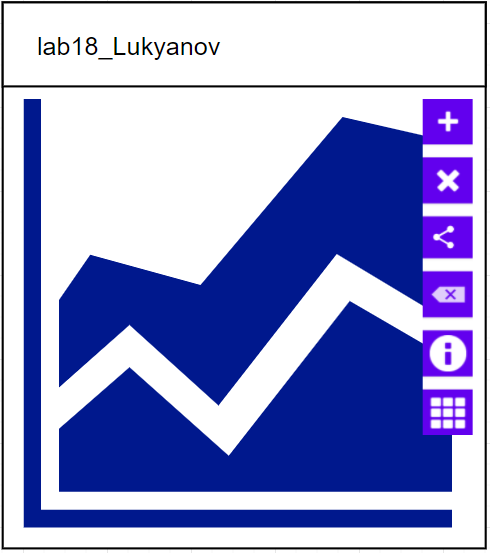


Рисунок 5 – Макет формы Main

На рисунке 6 показан внешний вид формы главного меню в приложении.

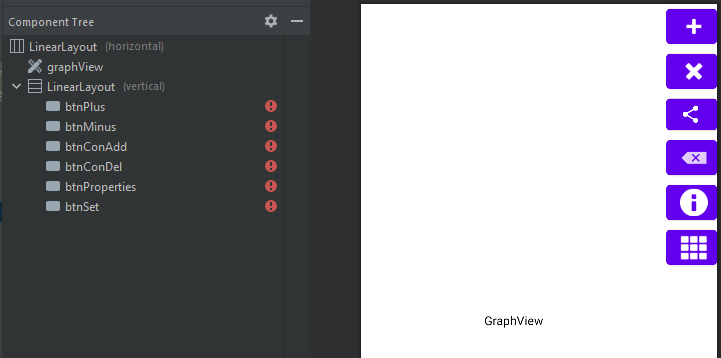


Рисунок 6 – Форма Main Menu в приложении

**Форма Graph**

На рисунке 7 показан макет внешнего вида формы графов.

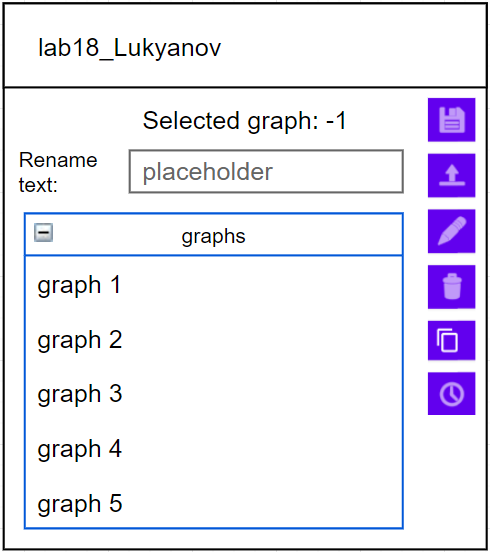


Рисунок 7 – Макет формы Graph

На рисунке 8 показан внешний вид формы графов в приложении.

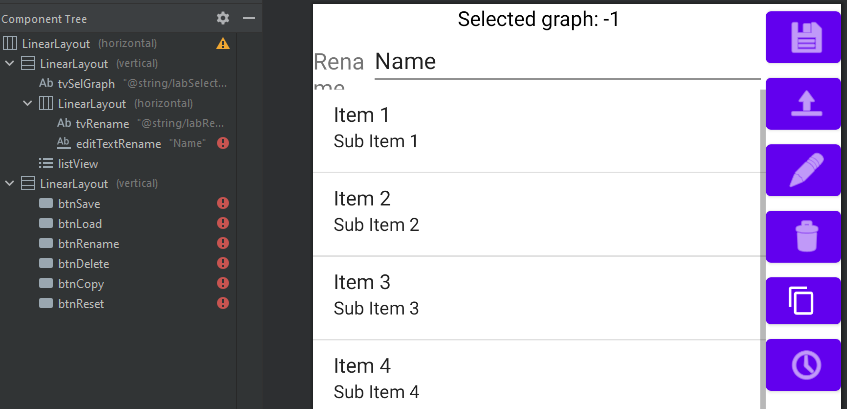


Рисунок 8 – Форма Graph в приложении

**Форма Node**

На рисунке 9 показан макет внешнего вида формы узла.

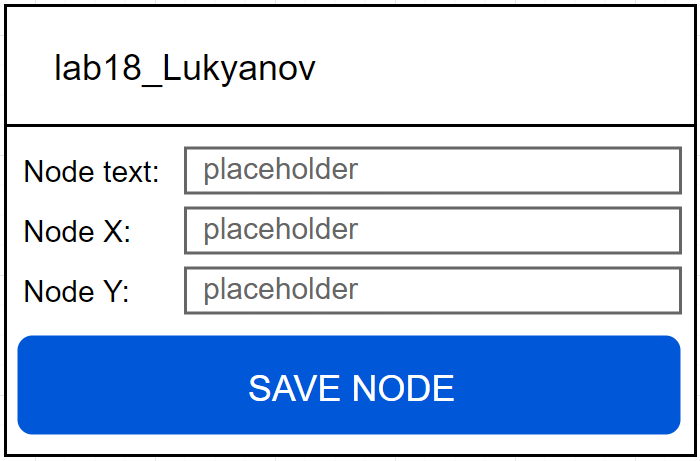


Рисунок 9 – Макет формы Node

На рисунке 10 показан внешний вид формы сообщения в приложении.

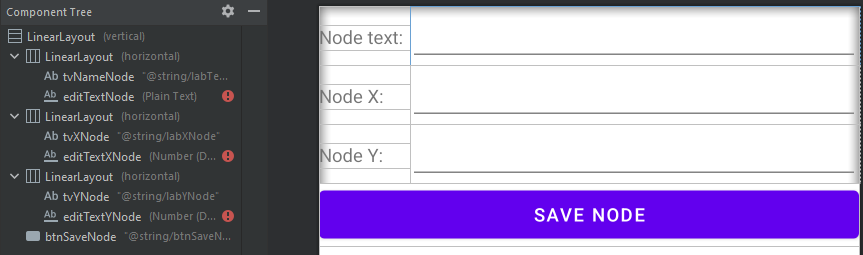


Рисунок 10 – Форма Node в приложении

**Форма Link**

На рисунке 11 показан макет внешнего вида формы связи.

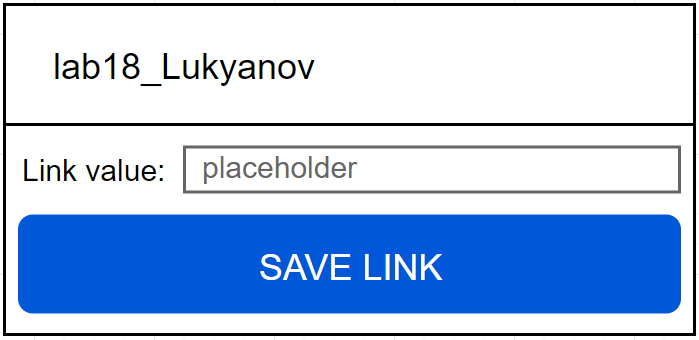


Рисунок 11 – Макет формы Link

На рисунке 12 показан внешний вид формы связи в приложении.

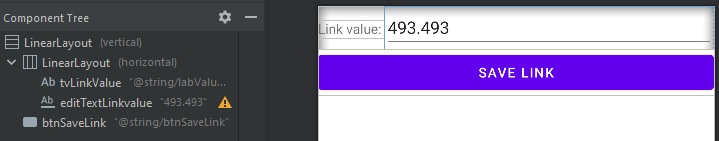


Рисунок 12 – Форма Link в приложении

Демонстрация работы приложения

Добавление узла представлено на рисунке 13:



Рисунок 13 – Добавление узла

Узел можно выбрать и переместить (рис 14):

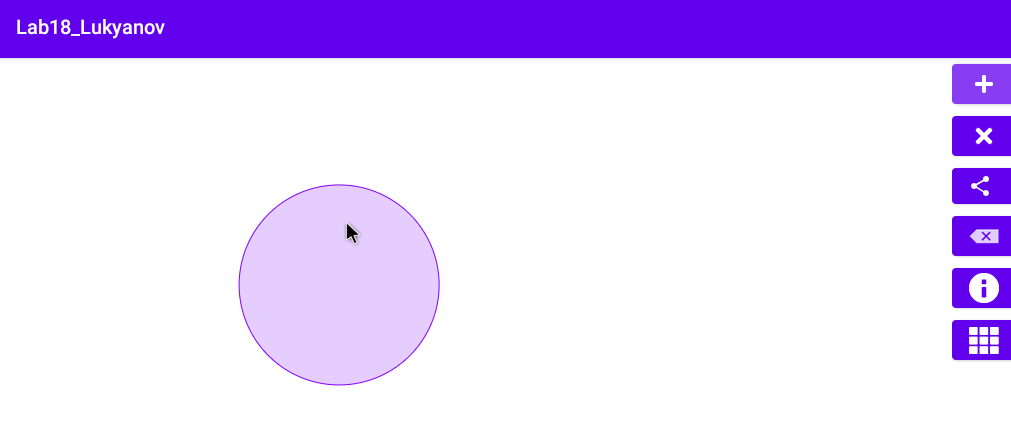


Рисунок 14 – Выбор и перемещение узла

Можно выбрать два узла (рис. 15):

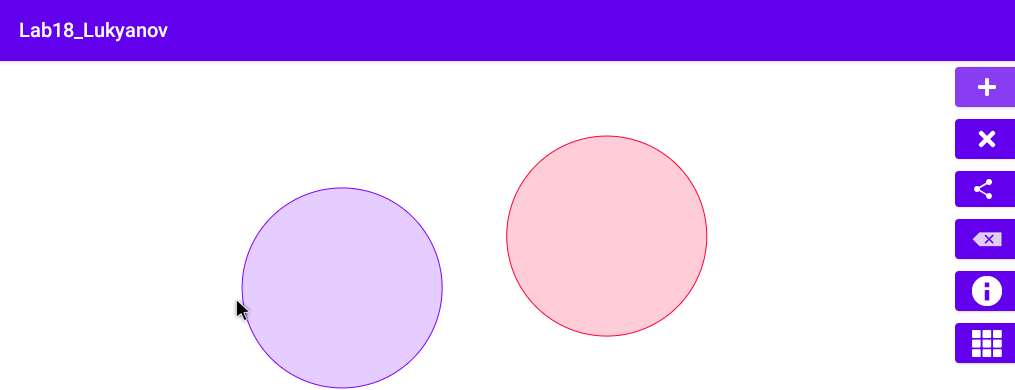


Рисунок 15 – Выбор двух узлов

Два узла можно связать, при этом открывается диалог с вводом значения связи (рис. 16):

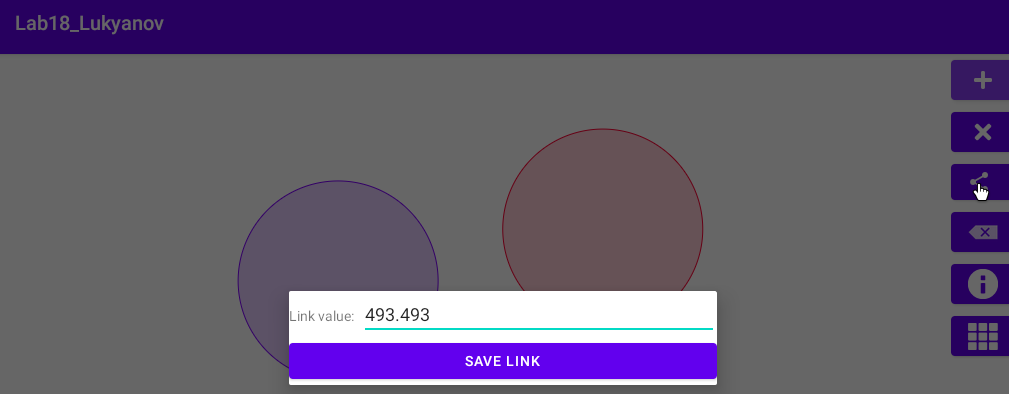


Рисунок 16 – Диалог для значения связи

Образуется связь с направлением (рис. 17):

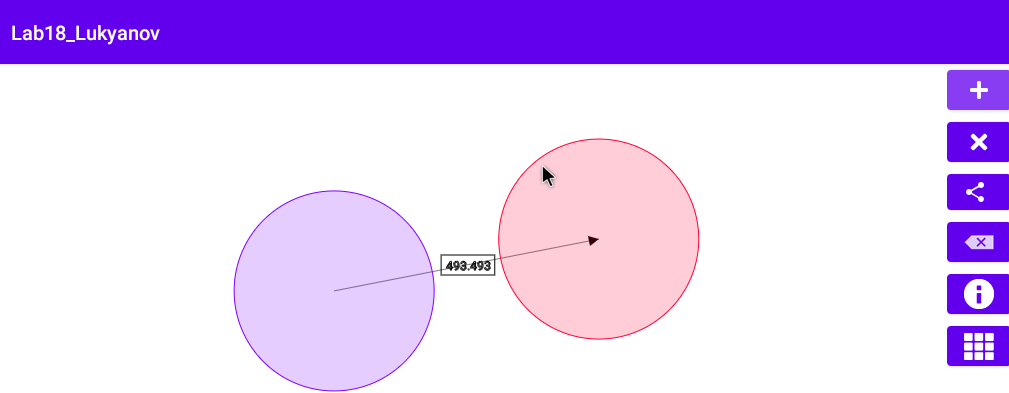


Рисунок 17 – Связь на графе

Двойная связь между узлами (рис. 18):

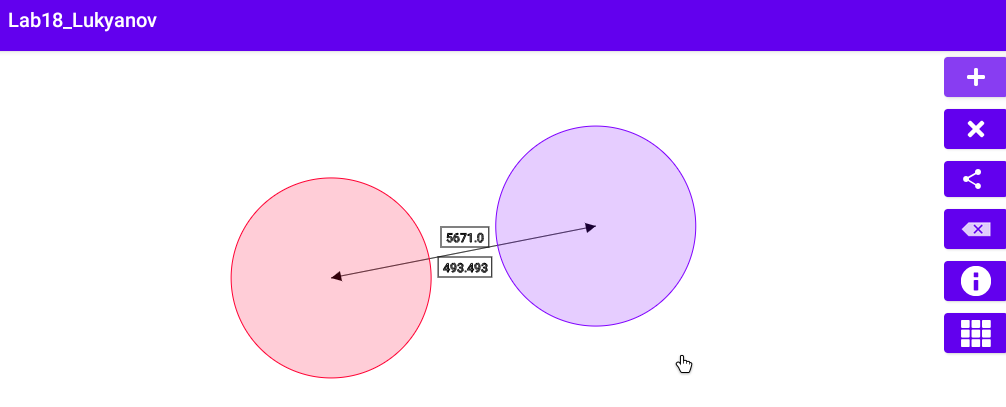


Рисунок 18 – Двойная связь

Связи можно выбрать и изменить (рис. 19):

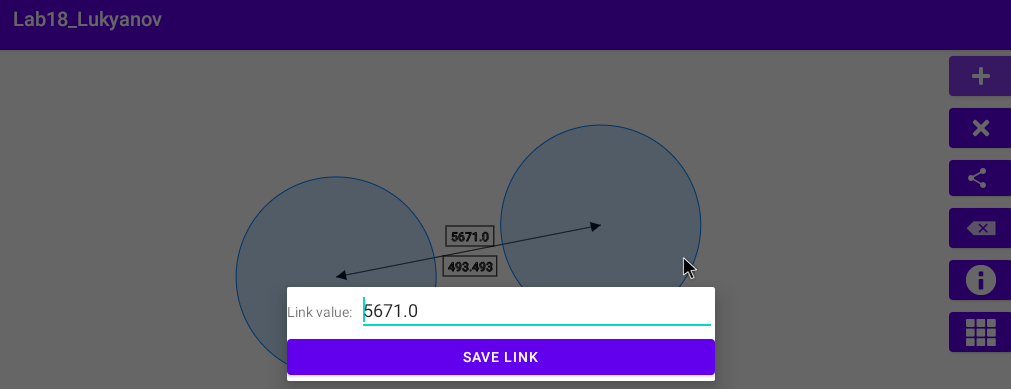


Рисунок 19 – Выбор и изменение значения связи

После изменения значения на «2022» (рис. 20):

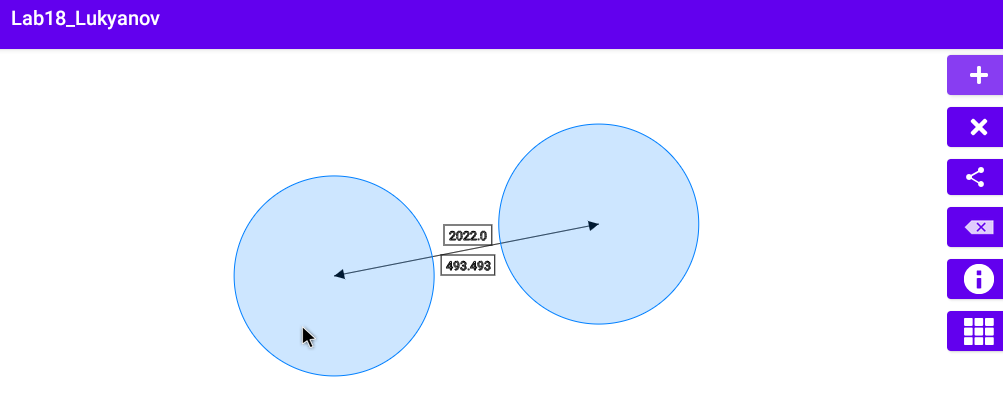


Рисунок 20 – Изменённое значение связи

Связь можно удалить, удалим связь «2022.0» (рис. 21):

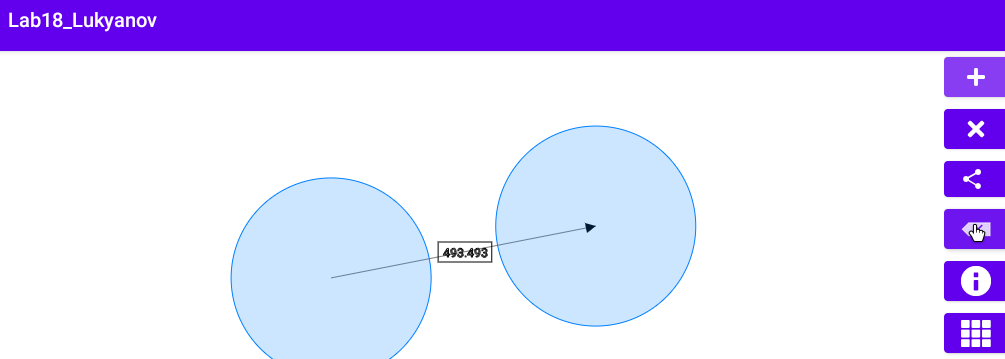


Рисунок 21 – Удаление связи

Узлы можно редактировать: задать имя и изменить координаты (рис. 22):

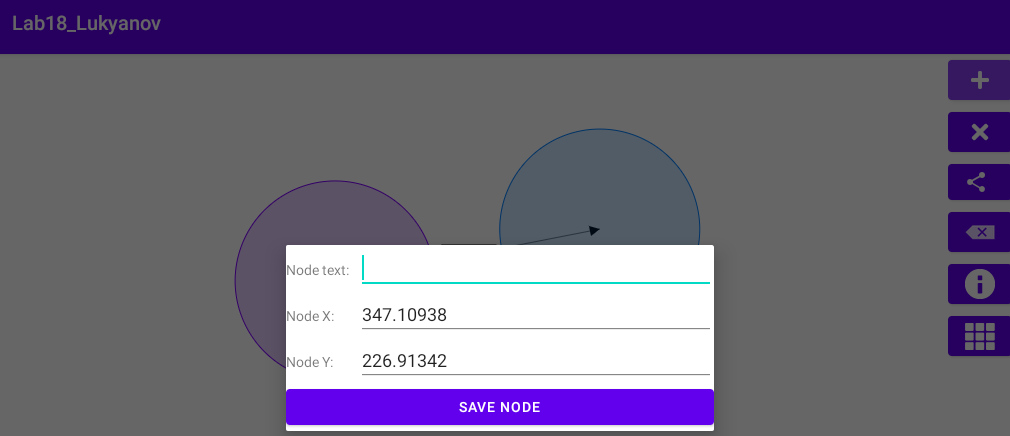


Рисунок 22 – Редактирования узла

Изменённые свойства (рис. 23):

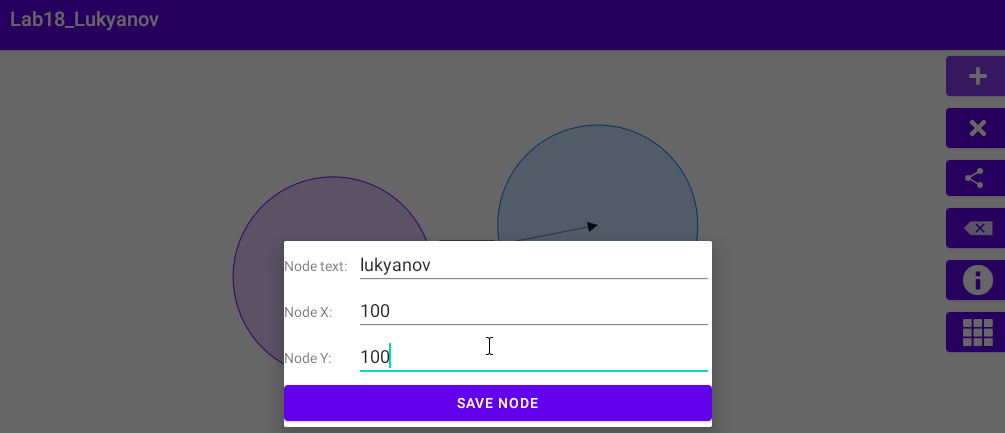


Рисунок 23 – Отредактированный узел

Узел после изменения (рис. 24):

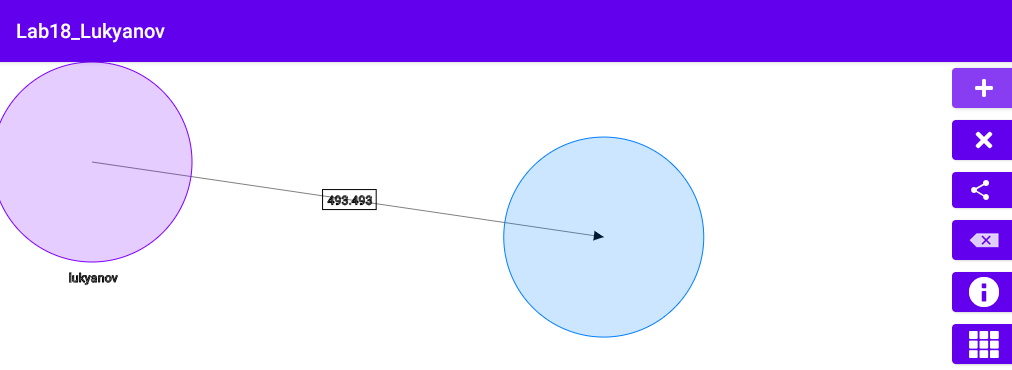


Рисунок 24 – Отредактированный узел на графе

После удаления узла удаляются его связи (рис. 25 и 26):

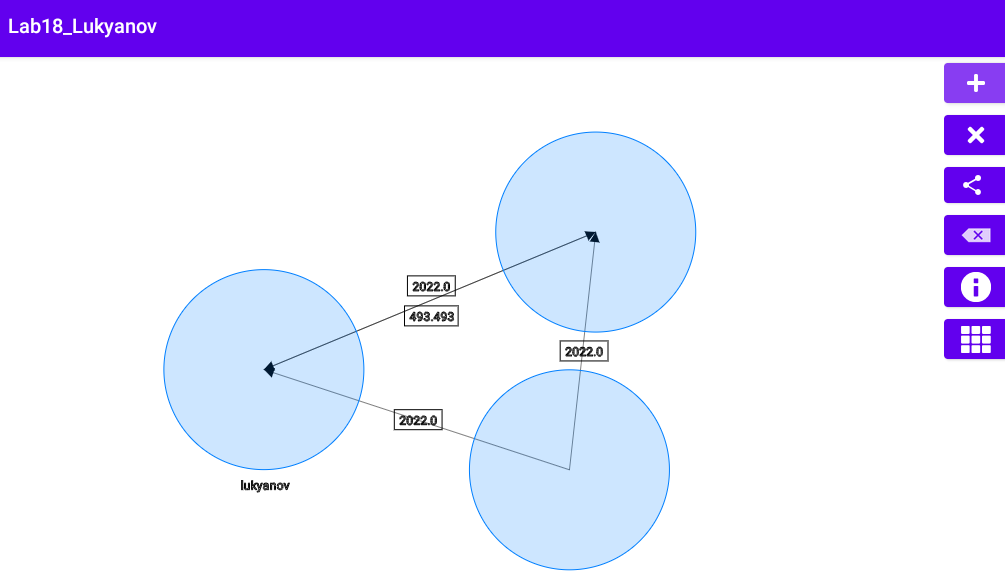


Рисунок 25 – Граф до удаления узла

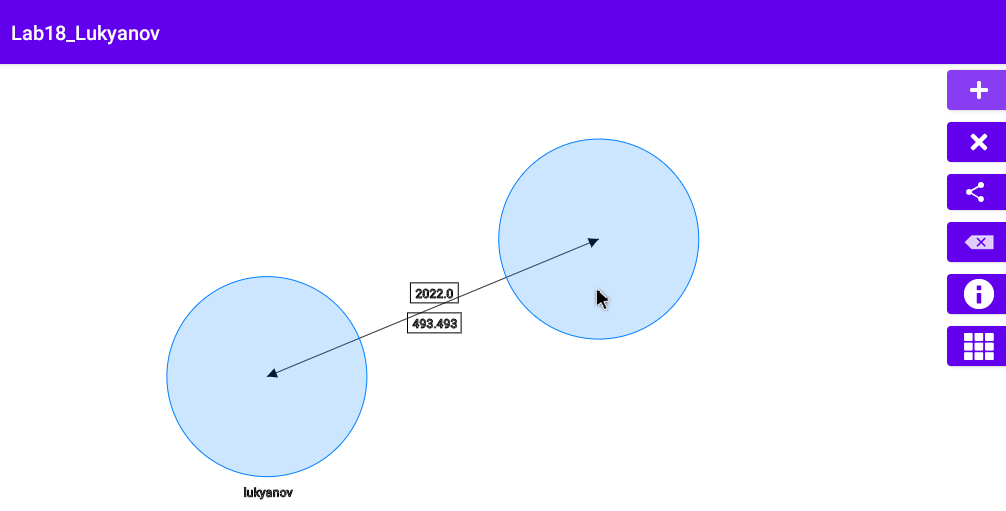


Рисунок 26 – Граф после удаления узла

Создадим граф для сохранения (рис. 27):

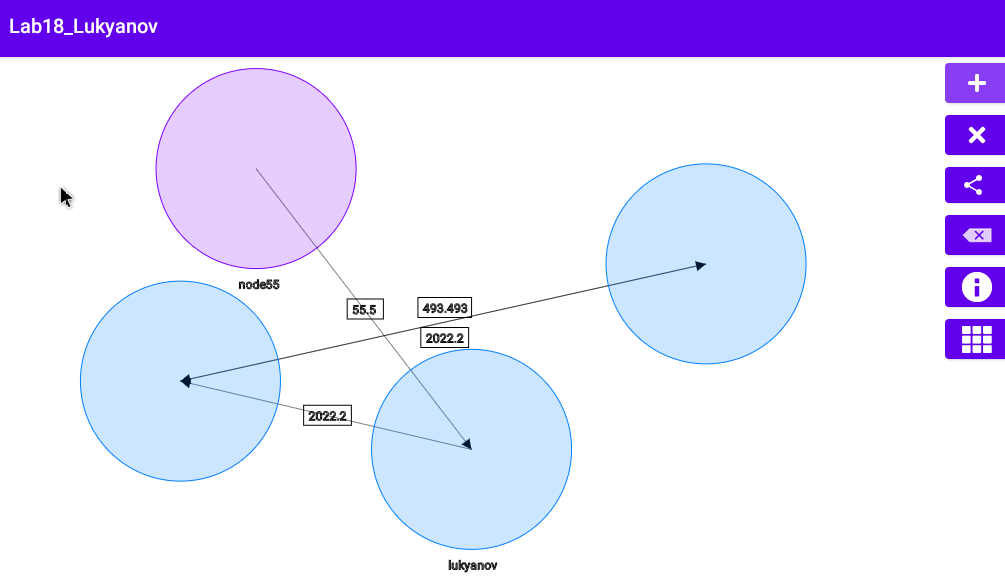


Рисунок 27 – Граф для сохранения

При нажатии на последнюю кнопку открывается форма с графами, они загружаются из базы данных (рис. 28):

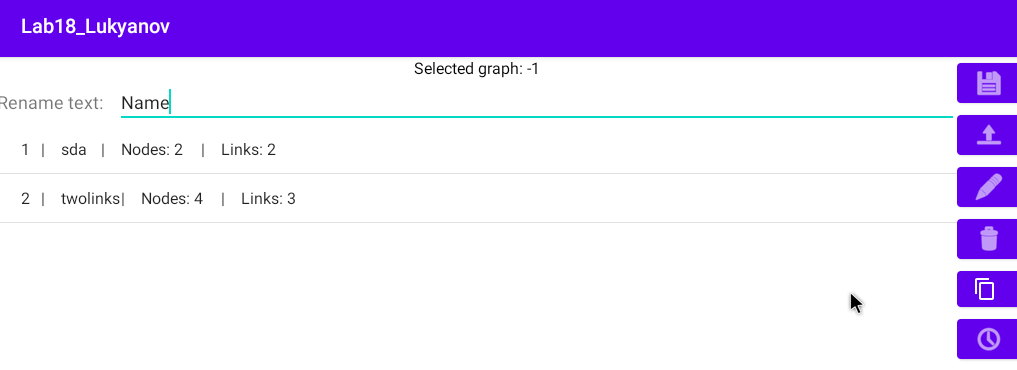


Рисунок 28 – Форма графов с сохраненными в базе графами

Сохраним граф с именем «lukyanov493» (рис. 29):

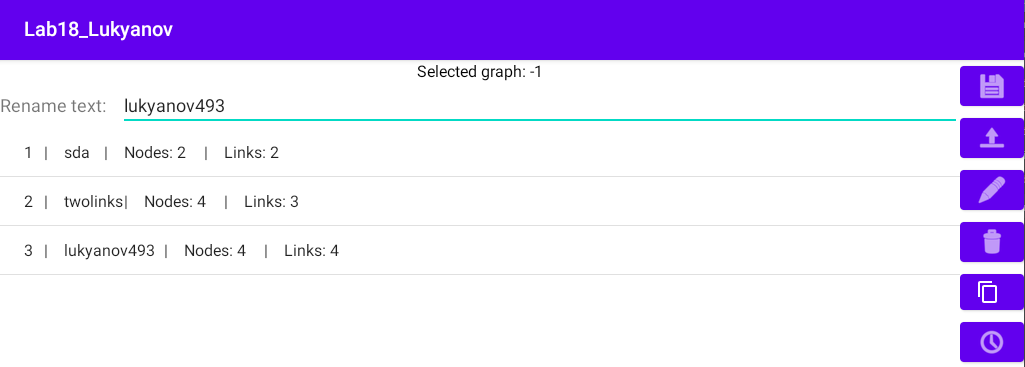


Рисунок 29 – Сохранение графа

Скопируем наш граф (рис. 30):



Рисунок 30 – Копирование графа

Переименуем копию на «lukyanov493\_copy», до этого необходимо выбрать граф (рис. 31):

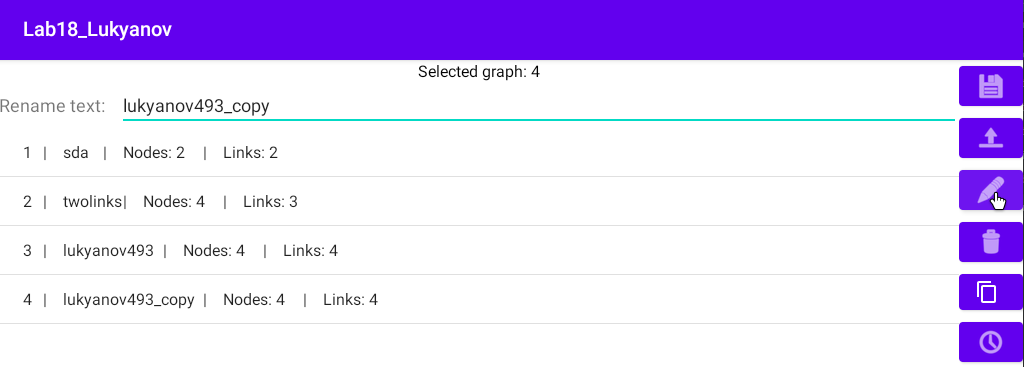


Рисунок 31 – Переименование графа

Загрузим граф с именем «sda» (рис. 32):

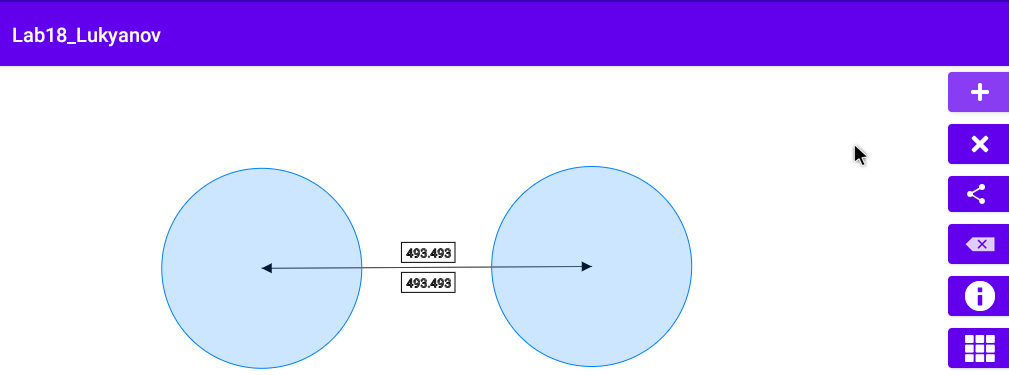


Рисунок 32 – Загрузка графа

Загрузим наш граф, он представлен на рисунке 27 (рис. 33):

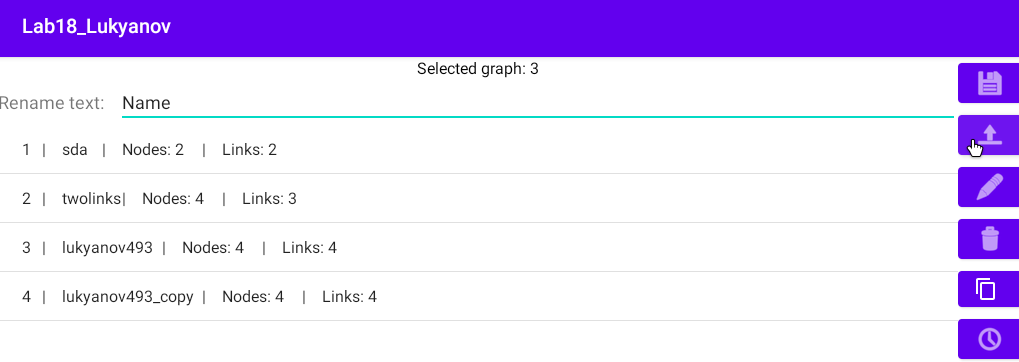


Рисунок 33 – Загрузка графа с рисунка 27

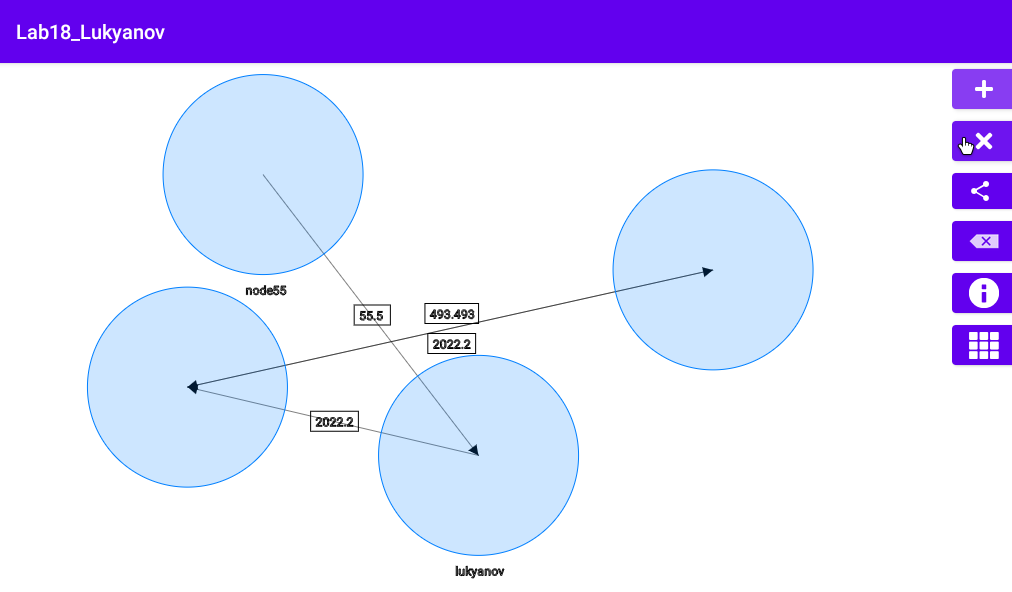


Рисунок 34 – Загруженный граф

С графом можно дальше работать (рис. 35):

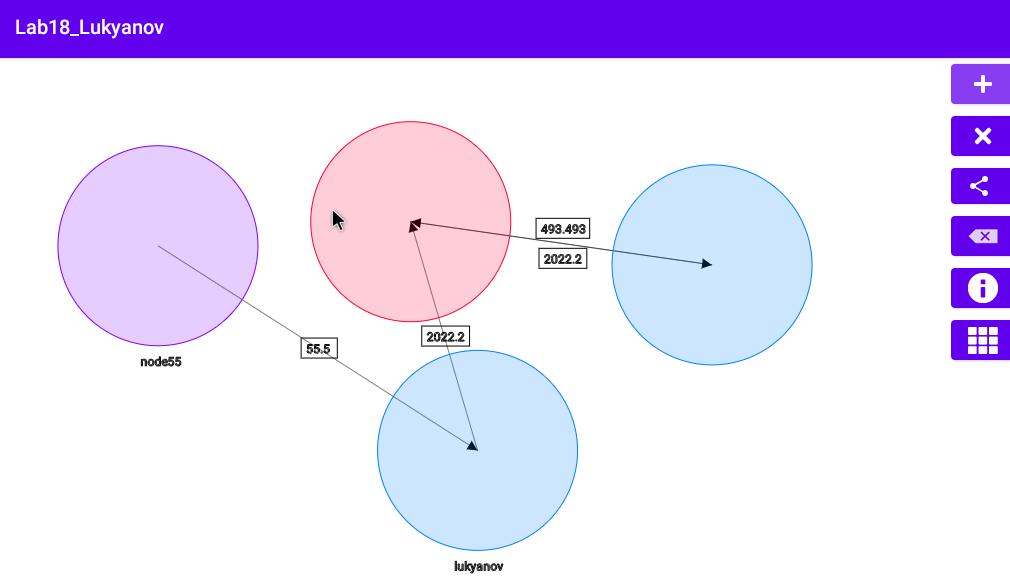


Рисунок 35 – Продолжение работы с графом

Также приложение имеет собственную иконку (рис. 36):

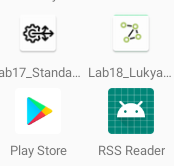


Рисунок 36 – Иконка приложения

Код приложения

**Класс «Node»»:**

package com.example.testlab\_18;  
  
public class Node  
{  
 public float x, y;  
 public String text;  
  
 public Node(float x, float y, String text)  
 {  
 this.x = x;  
 this.y = y;  
 this.text = text;  
 }  
}

**Класс «Link»»:**

package com.example.testlab\_18;  
  
public class Link  
{  
 public int a, b;  
 public float value;  
  
 public Link(int a, int b, float value)  
 {  
 this.a = a;  
 this.b = b;  
 this.value = value;  
 }  
}

**Класс «Graph»»:**

package com.example.testlab\_18;  
  
import java.util.ArrayList;  
  
public class Graph  
{  
 public String name;  
 public int number;  
  
 public ArrayList <Node> node = new ArrayList <Node> ();  
 public ArrayList <Link> link = new ArrayList <Link> ();  
  
 public void add\_node(float x, float y)  
 {  
 node.add(new Node(x, y, ""));  
 }  
 public void add\_link(int a, int b, float value)  
 {  
 link.add(new Link(a, b, value));  
 }  
  
 public void remove\_node(int index)  
 {  
 if (index < 0) return;  
 node.remove(index);  
 }  
  
 public void remove\_link(int index)  
 {  
 if (index < 0) return;  
 link.remove(index);  
 }  
  
 public String toString()  
 {  
 return number + "\t|\t" + name + "\t|\tNodes: " + node.size() + "\t|\tLinks: " + link.size();  
 }  
}

**Класс «DB»»:**

package com.example.testlab\_18;  
  
import android.content.Context;  
import android.database.Cursor;  
import android.database.sqlite.SQLiteDatabase;  
import android.database.sqlite.SQLiteOpenHelper;  
  
import androidx.annotation.Nullable;  
  
import java.util.ArrayList;  
  
public class DB extends SQLiteOpenHelper {  
 public DB(@Nullable Context context, @Nullable String name, @Nullable SQLiteDatabase.CursorFactory factory, int version) {  
 super(context, name, factory, version);  
 }  
  
 @Override  
 public void onCreate(SQLiteDatabase db) {  
 String sql = "CREATE TABLE Graph (number INT, name TEXT);";  
 db.execSQL(sql);  
 sql = "CREATE TABLE Node (graph INT, number INT, x REAL, y REAL, text TEXT);";  
 db.execSQL(sql);  
 sql = "CREATE TABLE Link (graph INT, number INT, a INT, b INT, value REAL);";  
 db.execSQL(sql);  
 }  
  
 public void onSaveGraph(Graph graph)  
 {  
 SQLiteDatabase db = getWritableDatabase();  
 int graphID = getMaxId("Graph");  
 graphID++;  
 graph.number = graphID;  
 int nodeID = getMaxId("Node");  
 nodeID++;  
 int linkID = getMaxId("Link");  
 linkID++;  
 String sql;  
 sql = "INSERT INTO Graph VALUES(" + graphID + ", '" + graph.name + "');";  
 db.execSQL(sql);  
 for (int i = 0; i < graph.node.size(); i++)  
 {  
 sql = "INSERT INTO Node VALUES (" + graphID + ", " + nodeID + ", " + graph.node.get(i).x + ", " + graph.node.get(i).y + ", '" + graph.node.get(i).text + "');";  
 db.execSQL(sql);  
 nodeID++;  
 }  
 for (int i = 0; i < graph.link.size(); i++)  
 {  
 sql = "INSERT INTO Link VALUES (" + graphID + ", " + linkID + ", " + graph.link.get(i).a + ", " + graph.link.get(i).b + ", " + graph.link.get(i).value + ");";  
 db.execSQL(sql);  
 linkID++;  
 }  
 }  
  
 public void onClearGraphs()  
 {  
 SQLiteDatabase db = getWritableDatabase();  
 String sql = "DELETE FROM Graph;";  
 db.execSQL(sql);  
 sql = "DELETE FROM Node;";  
 db.execSQL(sql);  
 sql = "DELETE FROM Link;";  
 db.execSQL(sql);  
 }  
  
 public int getMaxId(String table)  
 {  
 SQLiteDatabase db = getReadableDatabase();  
 String sql = "SELECT MAX(number) from " + table;  
 Cursor cur = db.rawQuery(sql, null);  
 if (cur.moveToFirst()) return cur.getInt(0);  
 return -1;  
 }  
  
 public void onRenameGraph(String name, int number)  
 {  
 if (number < 0) return;  
 SQLiteDatabase db = getWritableDatabase();  
 String sql = "UPDATE Graph SET name = '" + name + "' WHERE number = " + number + ";";  
 db.execSQL(sql);  
 }  
  
 public void onDeleteGraph(int number)  
 {  
 if (number < 0) return;  
 SQLiteDatabase db = getWritableDatabase();  
 String sql = "DELETE FROM Graph WHERE number = " + number + ";";  
 db.execSQL(sql);  
 sql = "DELETE FROM Node WHERE graph = " + number + ";";  
 db.execSQL(sql);  
 sql = "DELETE FROM Link WHERE graph = " + number + ";";  
 db.execSQL(sql);  
 }  
  
 public void getAllGraphs(ArrayList<Graph> lst)  
 {  
 SQLiteDatabase db = getReadableDatabase();  
 String sql = "SELECT \* FROM Graph;";  
 Cursor cur = db.rawQuery(sql, null);  
 Cursor curGraph;  
 if (cur.moveToFirst())  
 {  
 do {  
 Graph graph = new Graph();  
 graph.number = cur.getInt(0);  
 graph.name = cur.getString(1);  
 sql = "SELECT \* FROM Node WHERE graph = " + graph.number + ";";  
 curGraph = db.rawQuery(sql, null);  
 if (curGraph.moveToFirst())  
 {  
 do {  
 graph.add\_node(curGraph.getFloat(2), curGraph.getFloat(3));  
 int ab = curGraph.getPosition();  
 graph.node.get(ab).text = curGraph.getString(4);  
 }  
 while (curGraph.moveToNext());  
 }  
 sql = "SELECT \* FROM Link WHERE graph = " + graph.number + ";";  
 curGraph = db.rawQuery(sql, null);  
 if (curGraph.moveToFirst())  
 {  
 do {  
 graph.add\_link(curGraph.getInt(2), curGraph.getInt(3), curGraph.getFloat(4));  
 }  
 while (curGraph.moveToNext());  
 }  
 lst.add(graph);  
 }  
 while (cur.moveToNext());  
 }  
 }  
  
  
 public Graph onLoadGraph(int graphID)  
 {  
 SQLiteDatabase db = getReadableDatabase();  
 Graph graph = new Graph();  
 String sql = "SELECT \* FROM Graph WHERE number = " + graphID + ";";  
 Cursor cur = db.rawQuery(sql, null);  
 Cursor curGraph;  
 if (cur.moveToFirst())  
 {  
 graph.name = cur.getString(1);  
 sql = "SELECT \* FROM Node WHERE graph = " + graphID + ";";  
 curGraph = db.rawQuery(sql, null);  
 if (curGraph.moveToFirst())  
 {  
 do {  
 graph.add\_node(curGraph.getFloat(2), curGraph.getFloat(3));  
 graph.node.get(curGraph.getPosition()).text = curGraph.getString(4);  
 }  
 while (curGraph.moveToNext());  
 }  
 sql = "SELECT \* FROM Link WHERE graph = " + graphID + ";";  
 curGraph = db.rawQuery(sql, null);  
 if (curGraph.moveToFirst())  
 {  
 do {  
 graph.add\_link(curGraph.getInt(2), curGraph.getInt(3), curGraph.getFloat(4));  
 }  
 while (curGraph.moveToNext());  
 }  
 }  
 return graph;  
 }  
  
 @Override  
 public void onUpgrade(SQLiteDatabase sqLiteDatabase, int i, int i1) {  
  
 }  
}

**Класс «g»»:**

package com.example.testlab\_18;  
  
public final class g  
{  
 static DB *graph*;  
}

**Класс «GraphView»»:**

package com.example.testlab\_18;  
  
import android.content.Context;  
import android.graphics.Canvas;  
import android.graphics.Color;  
import android.graphics.Matrix;  
import android.graphics.Paint;  
import android.graphics.Path;  
import android.provider.CalendarContract;  
import android.util.AttributeSet;  
import android.view.MotionEvent;  
import android.view.SurfaceView;  
import android.widget.ArrayAdapter;  
  
import java.util.ArrayList;  
  
public class GraphView extends SurfaceView {  
  
 public Graph g = new Graph();  
 Paint p;  
  
 public int selected1 = -1;  
 public int selected2 = -1;  
 int lasthit = -1;  
 int lastHitNodeAndLink = -1;  
 int selectedLink = -1;  
  
 float rad = 100.0f;  
 float halfside = 5.0f;  
  
 float last\_x;  
 float last\_y;  
  
 float sizeText, sizeValueLink;  
  
 boolean nodeLink;  
  
 public void edit\_selected\_node(String text, float x, float y)  
 {  
 if (selected1 < 0) return;  
 g.node.get(selected1).text = text;  
 g.node.get(selected1).x = x;  
 g.node.get(selected1).y = y;  
 }  
  
 public Node get\_selected\_node()  
 {  
 if (selected1 < 0) return null;  
 Node n = g.node.get(selected1);  
 return n;  
 }  
  
 public Link get\_selected\_link()  
 {  
 if (selectedLink < 0) return null;  
 Link l = g.link.get(selectedLink);  
 return l;  
 }  
  
 public void edit\_selected\_link(float value)  
 {  
 if (selectedLink < 0) return;  
 g.link.get(selectedLink).value = value;  
 selectedLink = -1;  
 }  
  
 public void add\_node()  
 {  
 g.add\_node(100.0f, 100.0f);  
 invalidate();  
 }  
  
 public void remove\_selected\_node()  
 {  
 if (selected1 < 0) return;  
 g.remove\_node(selected1);  
 remove\_links\_at\_node(selected1);  
 selected1 = -1;  
 invalidate();  
 }  
  
 public void link\_selected\_nodes(float value)  
 {  
 if (selected1 < 0) return;  
 if (selected2 < 0) return;  
 if (check\_link\_exist(selected1, selected2))  
 {  
 g.add\_link(selected1, selected2, value);  
 selectedLink = -1;  
 invalidate();  
 }  
 }  
  
 public void remove\_selected\_link()  
 {  
 if (selectedLink < 0) return;  
 g.remove\_link(selectedLink);  
 selectedLink = -1;  
 invalidate();  
 }  
  
  
 public GraphView(Context context, AttributeSet attrs)  
 {  
 super(context, attrs);  
 p = new Paint();  
 p.setAntiAlias(true);  
 setWillNotDraw(false);  
 }  
  
 @Override  
 public boolean onTouchEvent(MotionEvent event) {  
 int action = event.getAction();  
 float x = event.getX();  
 float y = event.getY();  
 switch (action)  
 {  
 case MotionEvent.*ACTION\_DOWN*:  
 int i = get\_node\_at\_xy(x, y);  
 lasthit = i;  
 lastHitNodeAndLink = lasthit;  
 if (lastHitNodeAndLink < 0)  
 {  
 nodeLink = false;  
 lastHitNodeAndLink = get\_link\_at\_xy(x,y);  
 }  
 else nodeLink = true;  
 if (i < 0)  
 {  
 selected1 = -1;  
 selected2 = -1;  
 }  
 else  
 {  
 if (selected1 >= 0) selected2 = i;  
 else selected1 = i;  
 }  
 selectedLink = get\_link\_at\_xy(x, y);  
 last\_x = x;  
 last\_y = y;  
 invalidate();  
 return true;  
  
 case MotionEvent.*ACTION\_UP*:  
 break;  
  
 case MotionEvent.*ACTION\_MOVE*:  
 {  
 if (lasthit >= 0 && nodeLink)  
 {  
 Node n = g.node.get(lasthit);  
 n.x += x - last\_x;  
 n.y += y - last\_y;  
 invalidate();  
 }  
 last\_x = x;  
 last\_y = y;  
 return true;  
 }  
 }  
 return super.onTouchEvent(event);  
 }  
  
 public float calculateAngle(float x0, float y0, float x1, float y1)  
 {  
 float angle = (float)Math.*toDegrees*(Math.*atan2*(x1 - x0, y1 - y0));  
  
 angle = angle + (float)Math.*ceil*(-angle / 360) \* 360;  
  
 return angle;  
 }  
  
 public class Rectangle  
 {  
 public int linkID;  
 public float x0;  
 public float x1;  
 public float y0;  
 public float y1;  
  
 public Rectangle(int linkID, float x0, float y0, float x1, float y1)  
 {  
 this.linkID = linkID;  
 this.x0 = x0;  
 this.x1 = x1;  
 this.y0 = y0;  
 this.y1 = y1;  
 }  
 }  
 Rectangle[] linkRectangles;  
 ArrayList<Integer> skip = new ArrayList<Integer>();  
  
 public boolean isLinkSkip(int id)  
 {  
 for (int i = 0; i < skip.size(); i++)  
 {  
 if (id == skip.get(i)) return false;  
 }  
 return true;  
 }  
  
 @Override  
 protected void onDraw(Canvas canvas) {  
 canvas.drawColor(Color.*rgb*(255, 255, 255));  
 boolean twoLines = false;  
 skip.clear();  
 linkRectangles = new Rectangle[g.link.size()];  
 for (int i = 0; i < g.link.size(); i++)  
 {  
  
 Link l = g.link.get(i);  
 Node na = g.node.get(l.a);  
 Node nb = g.node.get(l.b);  
 p.setColor(Color.*argb*(127, 0, 0, 0));  
 canvas.drawLine(na.x, na.y, nb.x, nb.y, p);  
  
 *//arrows* p.setStyle(Paint.Style.*FILL*);  
 p.setColor(Color.*rgb*(0,0,0));  
 float angle = calculateAngle(na.x, na.y, nb.x, nb.y);  
 angle = 180 - angle;  
 Path arrow\_path = new Path();  
  
 Matrix arrow\_matrix = new Matrix();  
  
 arrow\_matrix.postRotate(angle, nb.x, nb.y);  
  
 arrow\_path.moveTo(nb.x, nb.y);  
 arrow\_path.lineTo(nb.x - 5, nb.y + 10);  
 arrow\_path.moveTo(nb.x, nb.y);  
 arrow\_path.lineTo(nb.x + 5, nb.y + 10);  
 arrow\_path.lineTo(nb.x - (5), nb.y + 10);  
 arrow\_path.transform(arrow\_matrix);  
  
 canvas.drawPath(arrow\_path, p);  
 if (isLinkSkip(i))  
 {  
 float bx = (na.x + nb.x) \* 0.5f;  
 float by = (na.y + nb.y) \* 0.5f;  
 float x0 = bx - halfside;  
 float y0 = by - halfside;  
 float x1 = bx + halfside;  
 float y1 = by + halfside;  
  
 *//value and twolines* p.setStyle(Paint.Style.*STROKE*);  
 p.setColor(Color.*rgb*(0, 0, 0));  
 sizeValueLink = String.*valueOf*(l.value).length() \* 2.9f;  
 twoLines = false;  
 for (int j = i+1; j < g.link.size(); j++) {  
 if (g.link.get(i).a == g.link.get(j).b && g.link.get(i).b == g.link.get(j).a) {  
 sizeValueLink = String.*valueOf*(g.link.get(i).value).length() \* 2.9f;  
 canvas.drawText("" + g.link.get(i).value, x0 + halfside - sizeValueLink, y0 + halfside \* 2 + 15, p);  
 linkRectangles[i] = new Rectangle(i, x0 - sizeValueLink, y0 + 10, x1 + sizeValueLink + 3, y1 + 20);  
 canvas.drawRect(x0 - sizeValueLink, y0 + 10, x1 + sizeValueLink + 3, y1 + 20, p);  
 sizeValueLink = String.*valueOf*(g.link.get(j).value).length() \* 2.9f;  
 canvas.drawText("" + g.link.get(j).value, x0 + halfside - sizeValueLink, y0 + halfside \* 2 - 15, p);  
 linkRectangles[j] = new Rectangle(j, x0 - sizeValueLink, y0 - 20, x1 + sizeValueLink + 3, y1 - 10);  
 canvas.drawRect(x0 - sizeValueLink, y0 - 20, x1 + sizeValueLink + 3, y1 - 10, p);  
 twoLines = true;  
 *//skip = j;* skip.add(j);  
 }  
 }  
 if (!twoLines) {  
 canvas.drawText("" + l.value, x0 + halfside - sizeValueLink, y0 + halfside \* 2, p);  
 linkRectangles[i] = new Rectangle(i, x0 - sizeValueLink, y0 - 5, x1 + sizeValueLink + 3, y1 + 5);  
 canvas.drawRect(x0 - sizeValueLink, y0-5, x1 + sizeValueLink+3, y1+5, p);  
 }  
 }  
 }  
 for (int i = 0; i < g.node.size(); i++)  
 {  
 Node n = g.node.get(i);  
  
 p.setStyle(Paint.Style.*FILL*);  
  
 if (i == selected1) p.setColor(Color.*argb*(50, 127, 0, 255));  
 else if (i == selected2 ) p.setColor(Color.*argb*(50, 255, 0, 50));  
 else p.setColor(Color.*argb*(50, 0, 127, 255));  
  
 canvas.drawCircle(n.x, n.y, rad, p);  
  
 p.setStyle(Paint.Style.*STROKE*);  
  
 if (i == selected1) p.setColor(Color.*rgb*(127, 0, 255));  
 else if (i == selected2) p.setColor(Color.*rgb*(255, 0, 50));  
 else p.setColor(Color.*rgb*(0,127,255));  
  
 canvas.drawCircle(n.x, n.y, rad, p);  
  
 if (n.text != null && n.text != "" && !n.text.isEmpty())  
 {  
 p.setColor(Color.*rgb*(0,0,0));  
 sizeText = n.text.length()\*2.9f;  
 canvas.drawText(n.text, n.x-sizeText, n.y+rad+20, p);  
 }  
 }  
 *//super.onDraw(canvas);* }  
  
 public int get\_node\_at\_xy(float x, float y)  
 {  
 for (int i = g.node.size() - 1; i >= 0; i--)  
 {  
 Node n = g.node.get(i);  
 float dx = x - n.x;  
 float dy = y - n.y;  
 if (dx \* dx + dy \* dy <= rad \* rad) return i;  
 }  
 return -1;  
 }  
  
 public int get\_link\_at\_xy(float x, float y)  
 {  
 for (int i = 0; i < g.link.size(); i++)  
 {  
 float x0 = linkRectangles[i].x0;  
 float y0 = linkRectangles[i].y0;  
 float x1 = linkRectangles[i].x1;  
 float y1 = linkRectangles[i].y1;  
 if (x >= x0 && x <= x1 && y >= y0 && y <= y1) return i;  
 }  
 return -1;  
 }  
  
 public void remove\_links\_at\_node(int node)  
 {  
 for (int i = g.link.size()-1; i >= 0; i--)  
 {  
 if (g.link.get(i).a == node || g.link.get(i).b == node)  
 {  
 g.remove\_link(i);  
 }  
 }  
 for (int i = 0; i < g.link.size(); i++)  
 {  
 if (g.link.get(i).a > node) g.link.get(i).a--;  
 if (g.link.get(i).b > node) g.link.get(i).b--;  
 }  
 }  
  
 public boolean check\_link\_exist(int a, int b)  
 {  
 for (int i = 0; i < g.link.size(); i++)  
 {  
 if (g.link.get(i).a == a && g.link.get(i).b == b)  
 return false;  
 }  
 return true;  
 }  
}

**Класс «MainActivity»»:**

package com.example.testlab\_18;  
  
import androidx.annotation.Nullable;  
import androidx.appcompat.app.AlertDialog;  
import androidx.appcompat.app.AppCompatActivity;  
  
import android.content.Intent;  
import android.nfc.FormatException;  
import android.os.Bundle;  
import android.view.LayoutInflater;  
import android.view.View;  
import android.widget.Button;  
import android.widget.EditText;  
import android.widget.Toast;  
  
public class MainActivity extends AppCompatActivity {  
  
 GraphView gv;  
 Graph graph;  
  
 Intent i;  
  
 AlertDialog alertDialogNode;  
 AlertDialog alertDialogLink;  
  
 EditText txtTextNode;  
 EditText txtXNode;  
 EditText txtYNode;  
  
 EditText txtValueLink;  
  
 Button btnSaveNode;  
 Button btnSaveLink;  
  
 int selectedNode;  
  
 View dialogViewLink;  
  
 Float valueLink;  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity\_main*);  
 gv = findViewById(R.id.*graphView*);  
  
 *//dialogNodeProperties* LayoutInflater dialogLayoutNode = LayoutInflater.*from*(this);  
 View dialogViewNode = dialogLayoutNode.inflate(R.layout.*dialog\_properties*, null);  
 alertDialogNode = new AlertDialog.Builder(this).create();  
 alertDialogNode.setView(dialogViewNode);  
 txtTextNode = dialogViewNode.findViewById(R.id.*editTextNode*);  
 txtXNode = dialogViewNode.findViewById(R.id.*editTextXNode*);  
 txtYNode = dialogViewNode.findViewById(R.id.*editTextYNode*);  
 btnSaveNode = dialogViewNode.findViewById(R.id.*btnSaveNode*);  
 btnSaveNode.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 if (txtXNode.getText().toString().isEmpty() || txtYNode.getText().toString().isEmpty())  
 {  
 Toast.*makeText*(dialogViewNode.getContext(), "X or Y is empty", Toast.*LENGTH\_SHORT*).show();  
 return;  
 }  
 String text = txtTextNode.getText().toString();  
 Float x = Float.*parseFloat*(txtXNode.getText().toString());  
 Float y = Float.*parseFloat*(txtYNode.getText().toString());  
 if (x < 0 || y < 0)  
 {  
 Toast.*makeText*(dialogViewNode.getContext(), "X or Y is less than 0", Toast.*LENGTH\_SHORT*).show();  
 return;  
 }  
 gv.edit\_selected\_node(text, x, y);  
 alertDialogNode.cancel();  
 gv.invalidate();  
 }  
 });  
  
 *//dialogLinkProperties* LayoutInflater dialogLayoutLink = LayoutInflater.*from*(this);  
 dialogViewLink = dialogLayoutLink.inflate(R.layout.*dialog\_link\_value*, null);  
 alertDialogLink = new AlertDialog.Builder(this).create();  
 alertDialogLink.setView(dialogViewLink);  
 txtValueLink = dialogViewLink.findViewById(R.id.*editTextLinkvalue*);  
 btnSaveLink = dialogViewLink.findViewById(R.id.*btnSaveLink*);  
 btnSaveLink.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 if (txtValueLink.getText().toString().isEmpty())  
 {  
 Toast.*makeText*(dialogViewLink.getContext(), "Value is empty", Toast.*LENGTH\_SHORT*).show();  
 return;  
 }  
 valueLink = Float.*parseFloat*(txtValueLink.getText().toString());  
 if (linkEdit)  
 {  
 gv.edit\_selected\_link(valueLink);  
 linkEdit = false;  
 }  
 else  
 gv.link\_selected\_nodes(valueLink);  
 alertDialogLink.cancel();  
 gv.invalidate();  
 }  
 });  
 }  
  
 public void on\_add\_click(View v)  
 {  
 gv.add\_node();  
 }  
  
 public void on\_remove\_click(View v)  
 {  
 gv.remove\_selected\_node();  
 }  
  
 public void on\_link\_add(View v)  
 {  
 if (gv.selected1 < 0) return;  
 if (gv.selected2 < 0) return;  
 if (gv.check\_link\_exist(gv.selected1, gv.selected2))  
 {  
 alertDialogLink.show();  
 }  
 }  
  
 public void on\_link\_remove(View v)  
 {  
 gv.remove\_selected\_link();  
 }  
  
 public void on\_graphs(View v)  
 {  
 i = new Intent(this, GraphActivity.class);  
 Graph graph = gv.g;  
 int countNode = 0, countLink = 0;  
 for (int j = 0; j < graph.node.size(); j++)  
 {  
 i.putExtra("graph\_node\_" + j + "x", graph.node.get(j).x);  
 i.putExtra("graph\_node\_" + j + "y", graph.node.get(j).y);  
 i.putExtra("graph\_node\_" + j + "text", graph.node.get(j).text);  
 countNode++;  
 }  
 for (int j = 0; j < graph.link.size(); j++)  
 {  
 i.putExtra("graph\_link\_" + j + "a", graph.link.get(j).a);  
 i.putExtra("graph\_link\_" + j + "b", graph.link.get(j).b);  
 i.putExtra("graph\_link\_" + j + "value", graph.link.get(j).value);  
 countLink++;  
 }  
 i.putExtra("countNode", countNode);  
 i.putExtra("countLink", countLink);  
 startActivityForResult(i, 1);  
 }  
  
 @Override  
 protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data)  
 {  
 gv.selected1 = -1;  
 gv.selected2 = -1;  
 if (resultCode == 1000)  
 {  
 gv.g = new Graph();  
 int countNode, countLink;  
 int a, b;  
 float x, y, value;  
 String text;  
 countNode = data.getIntExtra("countNode", -1);  
 countLink = data.getIntExtra("countLink", -1);  
 for (int j = 0; j < countNode; j++)  
 {  
 x = data.getFloatExtra("graph\_node\_" + j + "x", -1);  
 y = data.getFloatExtra("graph\_node\_" + j + "y", -1);  
 text = data.getStringExtra("graph\_node\_" + j + "text");  
 gv.g.add\_node(x, y);  
 gv.g.node.get(j).text = text;  
 }  
 for (int j = 0; j < countLink; j++)  
 {  
 a = data.getIntExtra("graph\_link\_" + j + "a", -1);  
 b = data.getIntExtra("graph\_link\_" + j + "b", -1);  
 value = data.getFloatExtra("graph\_link\_" + j + "value", -1);  
 gv.g.add\_link(a, b, value);  
 }  
 gv.invalidate();  
 }  
 super.onActivityResult(requestCode, resultCode, data);  
 }  
  
 boolean linkEdit = false;  
  
 public void on\_node\_prop(View v)  
 {  
 if (gv.nodeLink) {  
 Node n = gv.get\_selected\_node();  
 if (n == null) return;  
 txtTextNode.setText(n.text);  
 txtXNode.setText("" + n.x);  
 txtYNode.setText("" + n.y);  
 alertDialogNode.show();  
 }  
 else  
 {  
 linkEdit = true;  
 Link l = gv.get\_selected\_link();  
 if (l == null) return;  
 txtValueLink.setText("" + l.value);  
 alertDialogLink.show();  
 }  
 }  
  
}

**Класс «GraphActivity»»:**

package com.example.testlab\_18;  
  
import androidx.appcompat.app.AppCompatActivity;  
  
import android.content.Intent;  
import android.os.Bundle;  
import android.view.View;  
import android.widget.ArrayAdapter;  
import android.widget.EditText;  
import android.widget.ListView;  
import android.widget.TextView;  
  
import java.util.ArrayList;  
  
public class GraphActivity extends AppCompatActivity {  
  
 ListView lstctl;  
 ArrayList<Graph> lst = new ArrayList<>();  
 ArrayAdapter<Graph> adp;  
  
 TextView tvSelectedGraph;  
 EditText txtName;  
  
 GraphView gv;  
 Graph graph;  
  
 Intent i;  
  
 int selectedGraph;  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.activity\_graph);  
  
 g.graph = new DB(this, "graph.db", null, 1);  
  
 tvSelectedGraph = findViewById(R.id.tvSelGraph);  
 txtName = findViewById(R.id.editTextRename);  
 lstctl = findViewById(R.id.listView);  
 lstctl.setOnItemClickListener((parent, view, position, id) ->  
 {  
 selectedGraph = (int) id + 1;  
 tvSelectedGraph.setText("Selected graph: " + selectedGraph);  
 });  
 adp = new ArrayAdapter<Graph>(this, android.R.layout.simple\_list\_item\_1, lst);  
 lstctl.setAdapter(adp);  
  
 updateList();  
  
 i = getIntent();  
 graph = new Graph();  
 int countNode, countLink;  
 String text;  
 int a, b;  
 float x, y, value;  
 countNode = i.getIntExtra("countNode", -1);  
 countLink = i.getIntExtra("countLink", -1);  
 for (int j = 0; j < countNode; j++)  
 {  
 x = i.getFloatExtra("graph\_node\_" + j + "x", -1);  
 y = i.getFloatExtra("graph\_node\_" + j + "y", -1);  
 text = i.getStringExtra("graph\_node\_" + j + "text");  
 graph.add\_node(x, y);  
 graph.node.get(j).text = text;  
 }  
 for (int j = 0; j < countLink; j++)  
 {  
 a = i.getIntExtra("graph\_link\_" + j + "a", -1);  
 b = i.getIntExtra("graph\_link\_" + j + "b", -1);  
 value = i.getFloatExtra("graph\_link\_" + j + "value", -1);  
 graph.add\_link(a, b, value);  
 }  
 }  
  
 public void updateList()  
 {  
 lst.clear();  
 g.graph.getAllGraphs(lst);  
 adp.notifyDataSetChanged();  
 }  
  
 public void onSave(View v)  
 {  
 graph.name = txtName.getText().toString();  
 g.graph.onSaveGraph(graph);  
 updateList();  
 }  
  
 public void onLoad(View v)  
 {  
 if (selectedGraph < 0) return;  
 graph = g.graph.onLoadGraph(selectedGraph);  
 int countNode = 0, countLink = 0;  
 for (int j = 0; j < graph.node.size(); j++)  
 {  
 i.putExtra("graph\_node\_" + j + "x", graph.node.get(j).x);  
 i.putExtra("graph\_node\_" + j + "y", graph.node.get(j).y);  
 i.putExtra("graph\_node\_" + j + "text", graph.node.get(j).text);  
 countNode++;  
 }  
 for (int j = 0; j < graph.link.size(); j++)  
 {  
 i.putExtra("graph\_link\_" + j + "a", graph.link.get(j).a);  
 i.putExtra("graph\_link\_" + j + "b", graph.link.get(j).b);  
 i.putExtra("graph\_link\_" + j + "value", graph.link.get(j).value);  
 countLink++;  
 }  
 i.putExtra("countNode", countNode);  
 i.putExtra("countLink", countLink);  
 setResult(1000, i);  
 finish();  
 }  
  
 public void onRename(View v)  
 {  
 g.graph.onRenameGraph(txtName.getText().toString(), selectedGraph);  
 updateList();  
 }  
  
 public void onDelete(View v)  
 {  
 g.graph.onDeleteGraph(selectedGraph);  
 updateList();  
 }  
  
 public void onCopy(View v)  
 {  
 g.graph.onSaveGraph(g.graph.onLoadGraph(selectedGraph));  
 updateList();  
 }  
  
 public void onReset(View v)  
 {  
 g.graph.onClearGraphs();  
 updateList();  
 }  
  
  
}