**Министерство образования и науки Российской Федерации**

Федеральное государственное автономное образовательное учреждение

высшего профессионального образования

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ**

**ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Инженерная школа информационных технологий и робототехники

Отделение информационных технологий

Направление – Информационные системы и технологии

**Лабораторная работа №3**

по дисциплине: Программирование мобильных устройств и встраиваемых систем

|  |  |  |  |
| --- | --- | --- | --- |
| Выполнил: студент гр. 8И6А | \_\_\_\_\_\_\_\_\_\_\_ | Н. П. Шкулов | |
|  |  | |  |
| Проверил: ассистент ОИТ | \_\_\_\_\_\_\_\_\_\_\_ | | А. В. Погребной |

Томск 2020 г.

# Задание

На экран со студентами добавить SearchView. При поиске искать по ФИО студентов в списке на наличие введённой фразы в SearchView и выводить отфильтрованный список. Для каждого результата подсвечивать цветом кусок текста, который нашёлся (через SpannableString).

# Ход работы

Согласно заданию, выделение цвета нужно осуществлять посредством SpannableString, поэтому в класс Student, имеющийся в примере выполнения работы, было добавлено соответствующее поле:

@NonNull

public String firstName;

@NonNull

public String secondName;

@NonNull

public String lastName;

@NonNull

public Spannable shownName;

Это новое поле представляет собой объеденное ФИО, к которому будут применяться текстовые стили (выделение цветом) и который будет выводиться в списке студентов.

В главном активити модуля этой лабораторной в функцию onCreateOptionsMenu() был добавлен элемент поиска, который будет выводиться в тулбаре.

MenuItem search = menu.findItem(R.id.search);

SearchView searchView = (SearchView) MenuItemCompat.getActionView(search);

search(searchView);

Сама функция search обрабатывает изменения текста в поле поиска и запрашивает применение фильтра через адаптер.

private void search(SearchView searchView) {

searchView.setOnQueryTextListener(new SearchView.OnQueryTextListener() {

//произошло нажатие "подтвердить"

@Override

public boolean onQueryTextSubmit(String query) {

studentsAdapter.getFilter().filter(query);

return false;

}

//произошло любое изменение текста

@Override

public boolean onQueryTextChange(String newText) {

studentsAdapter.getFilter().filter(newText);

return true;

}

});

}

В самом адаптере используется два списка студентов: один из них, students, хранит всех студентов, позволяя сбрасывать фильтр, тогда как другой, filteredStudents, получает отфильтрованные списки, которые и выводятся на экран.

private List<Student> students = new ArrayList<>();

private List<Student> filteredStudents = new ArrayList<>();

В самом фильтре в адаптере сначала происходит проверка на пустоту. Если введенный запрос пустой, то это значит, что либо пользователь только начал вводить запрос, либо, наоборот, он его стер. Соответственно, нужно убрать все текстовые выделения, так как фильтр отсутствует, и вернуть полный список студентов.

if (charString.isEmpty()) {

for (Student student : students) {

clearColor(student);

}

filteredStudents = students;

}

В противном случае происходит обход всех студентов, среди которых обнаруживаются подходящие запросу с помощью функции contains(). У всех найденных таким образом студентов сначала сбрасывается предыдущее выделение, а затем выполняется поиск, на каких именно отрезках строки было найдено совпадение. По найденным отрезкам накладываются новые выделения, после чего полученный отфильтрованный список передается на вывод.

else {

ArrayList<Student> filteredList = new ArrayList<>();

for (Student student : students) {

if (student.getName().toLowerCase().contains(charString)) {

clearColor(student);

List<Integer> positions = findWord(charString,

student.shownName.toString().toLowerCase());

for (int i = 0; i < positions.size()-1; i = i + 2)

{

if (positions.get(i) != -1)

student.shownName.setSpan(new ForegroundColorSpan(Color.RED), positions.get(i), positions.get(i+1)+1, Spanned.SPAN\_EXCLUSIVE\_EXCLUSIVE);

}

filteredList.add(student);

}

}

filteredStudents = filteredList;

}

После этого срабатывает функция publishResults, которая отвечает за обновление отфильтрованного выводимого списка и уведомление, что этот список изменился.

@Override

protected void publishResults(CharSequence charSequence,

FilterResults filterResults)

{

filteredStudents = (ArrayList<Student>) filterResults.values;

notifyDataSetChanged();

}

# Работа приложения

На главном экране приложения выводятся студенты, которые были добавлены изначально через модуль app, как показано на рисунке 1.

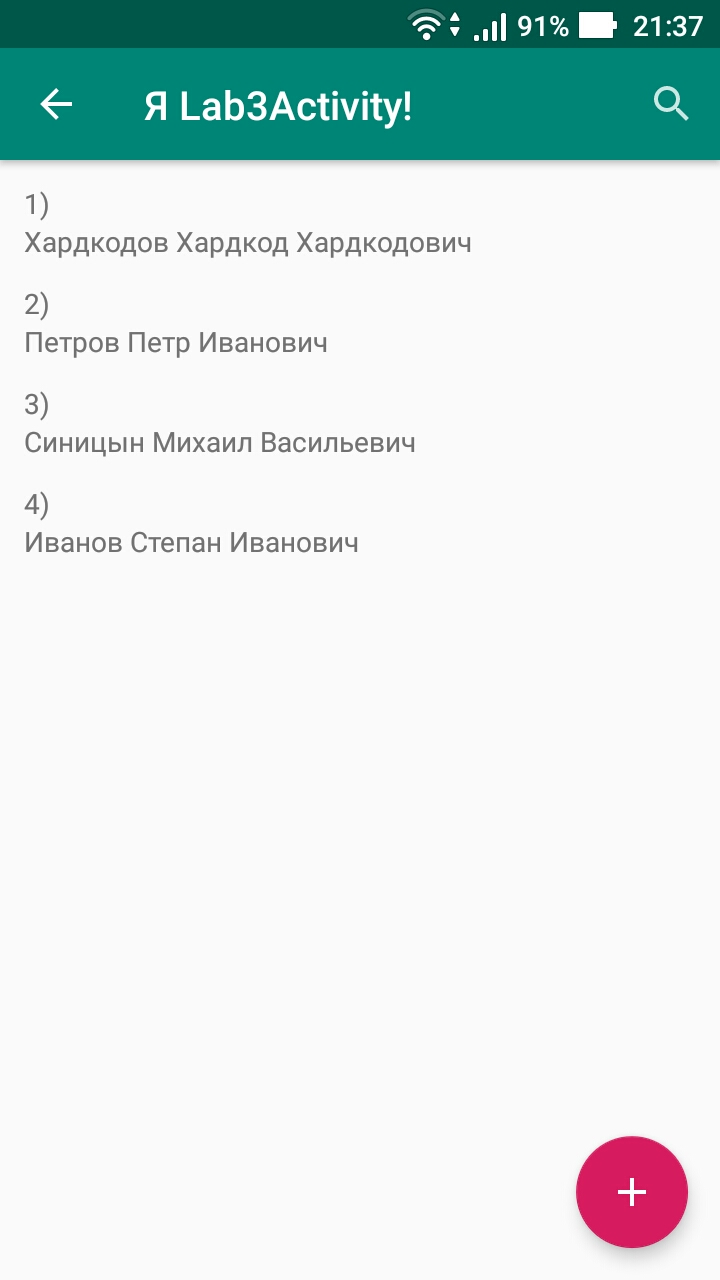


Рисунок 1 – Список студентов по умолчанию

При нажатии на кнопку в правом нижнем углу открывается экран для добавления студента, как показано на рисунке 2.

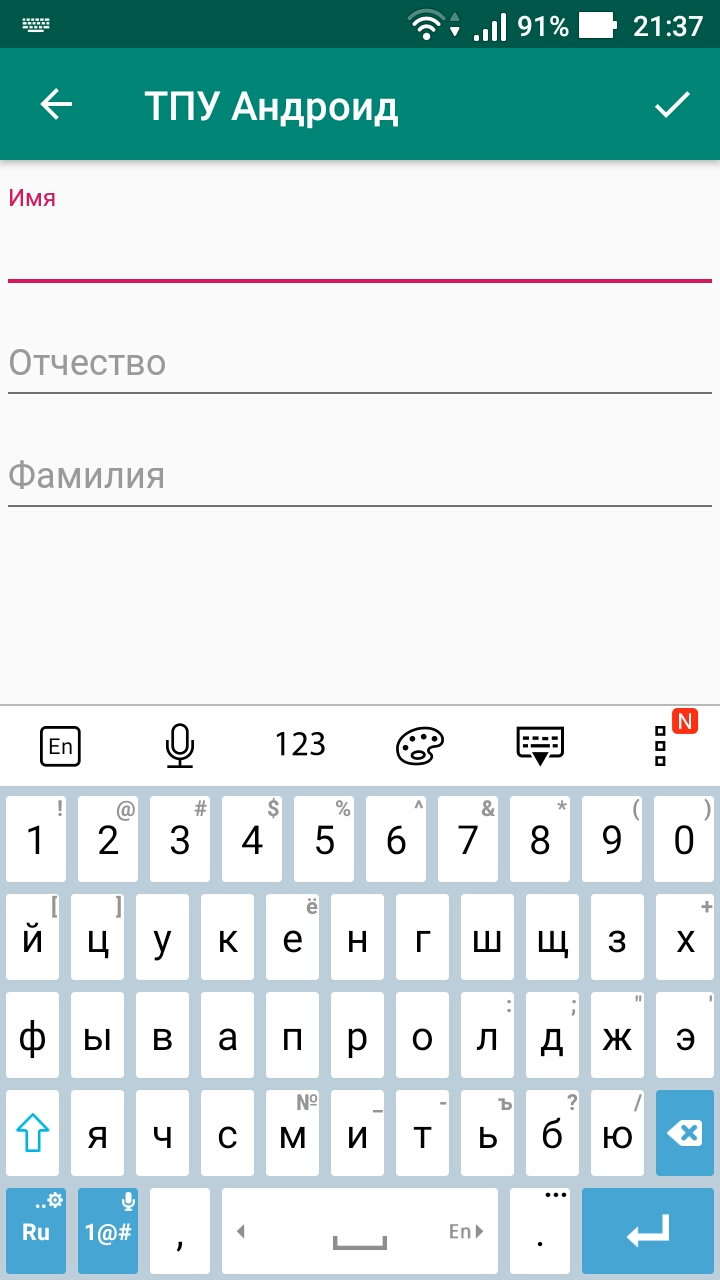


Рисунок 2 – Добавление нового студента

Так, был добавлен новый студент, как показано на рисунке 3.

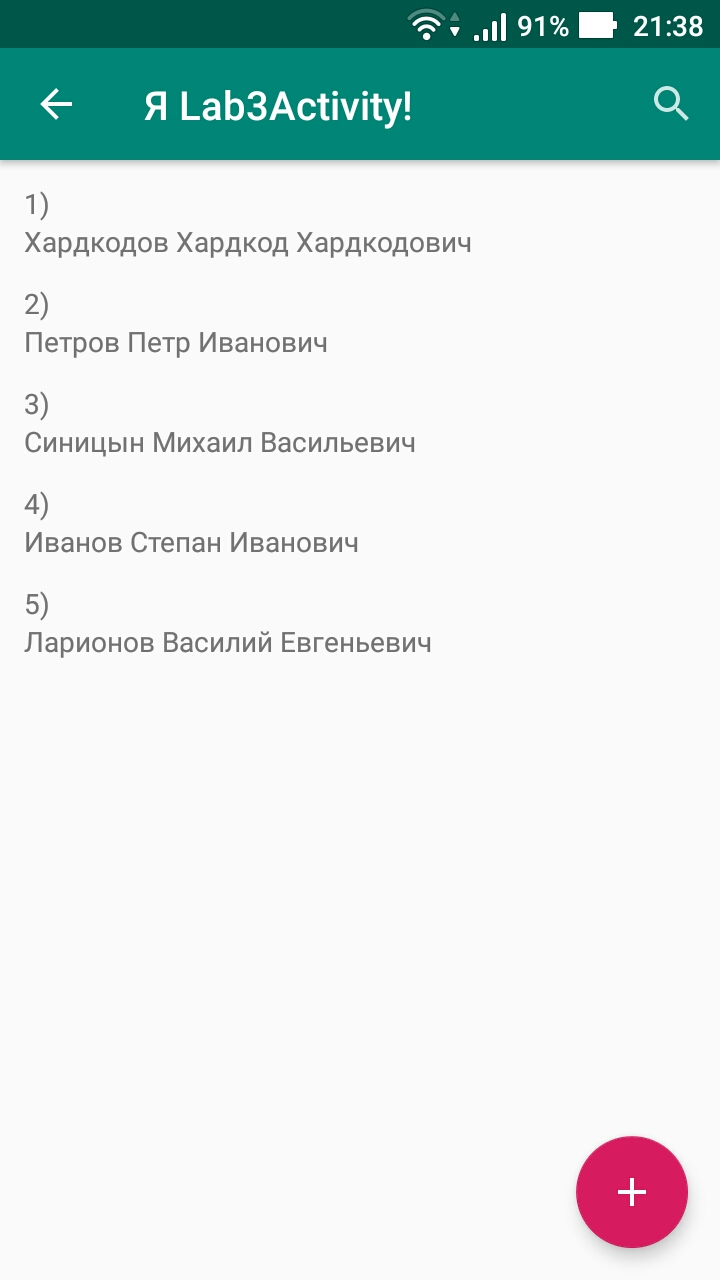


Рисунок 3 – Список с новым добавленным студентом

В тулбаре есть строка поиска, как можно видеть на рисунке 4.

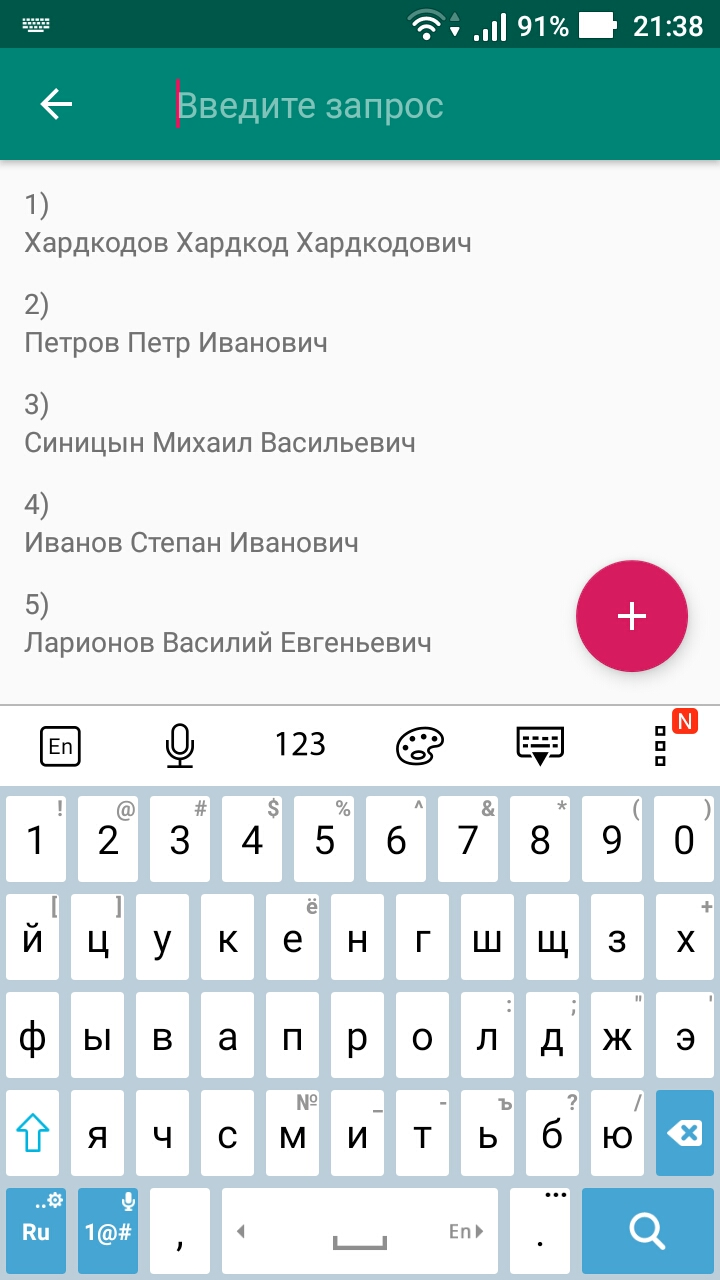


Рисунок 4 – Ввод запроса в строку поиска

На рисунках 5 и 6 показано, что при вводе запроса список автоматически отфильтровывается: на экране остаются только студенты, имеющие совпадения в ФИО с запросом, и эти совпадения окрашиваются в красный.

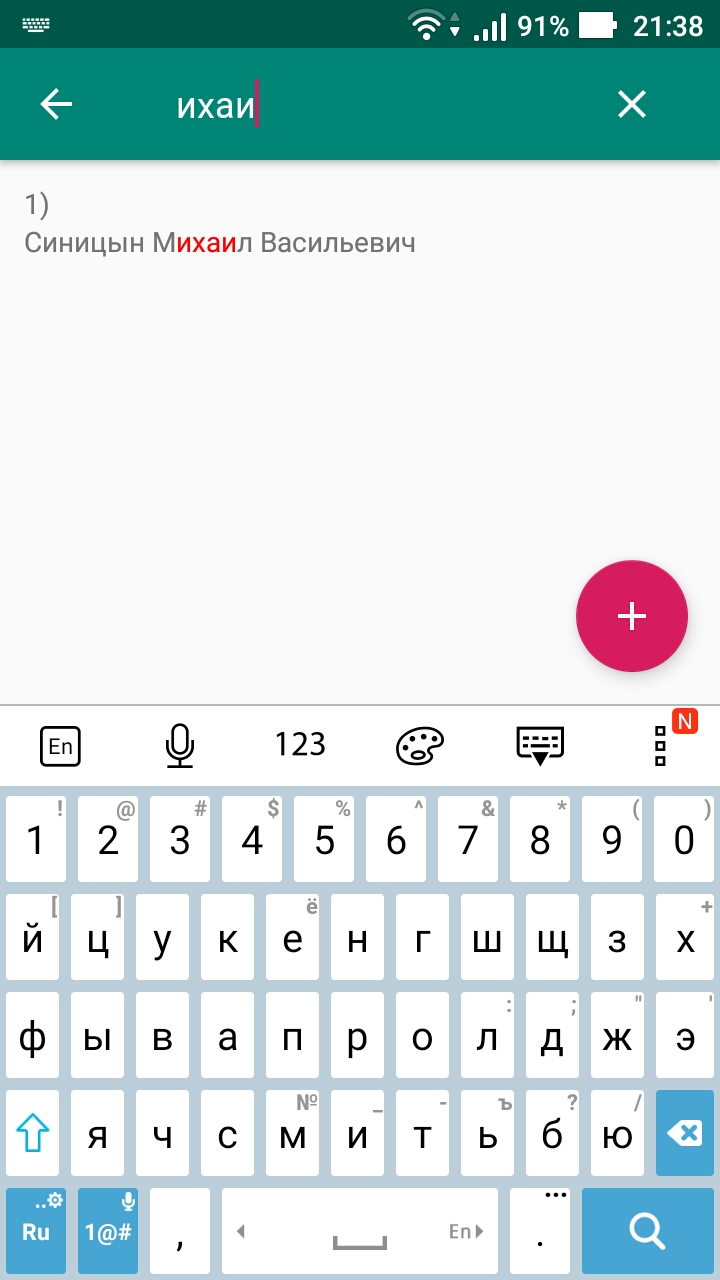


Рисунок 5 – Единичное совпадение

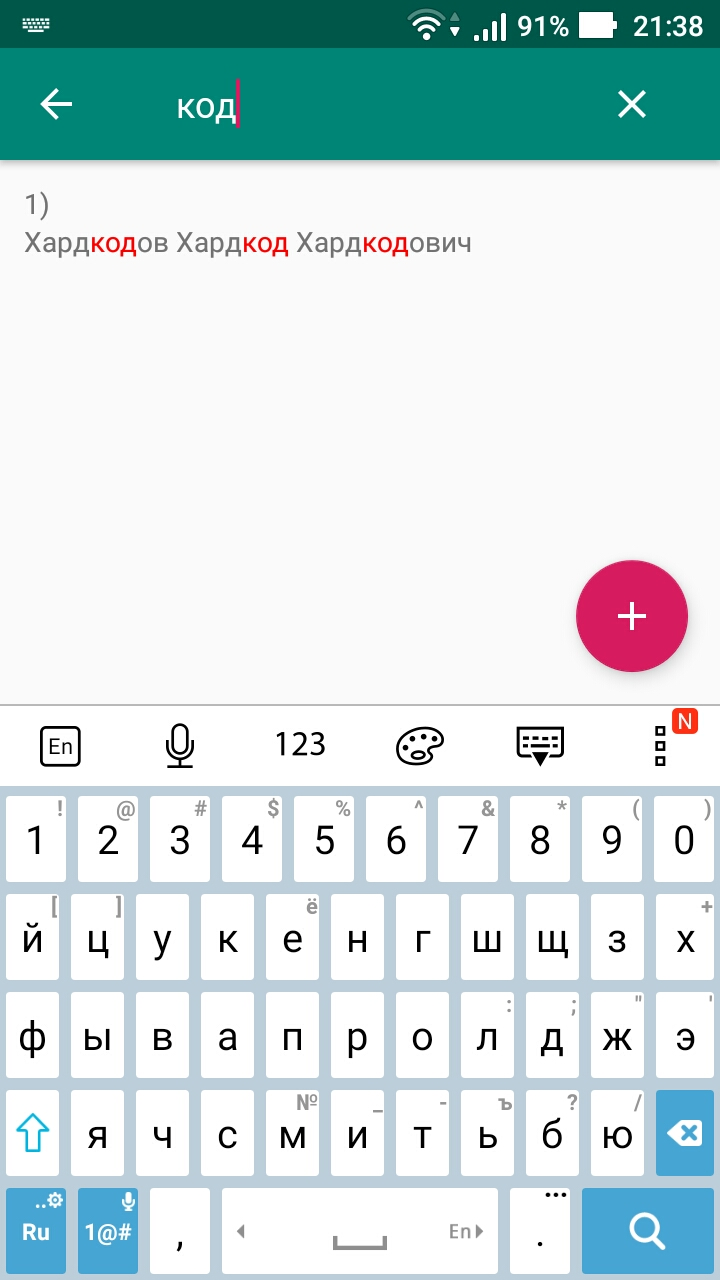


Рисунок 6 – Множество совпадений

# Итоги работы

По итогам выполнения лабораторной работы было получено приложение, хранящее список студентов, позволяющее добавлять новых и осуществлять поиск с подсвечиванием соответствий. Соответственно, с одной стороны, в данной лабораторной работе был рассмотрен способ хранения данных (а именно, посредством кэша), их вывода в RecyclerView (который пусть и сложнее в работе, чем обычный список, но производительнее за счет оптимизации вывода элементов списка), а также выделения строк с помощью специального типа данных Spannable.