BÁO CÁO THỰC HÀNH

Bài thực hành số 03: SQLite

Môn học: NT118 – Phát triển ứng dụng trên thiết bị di động

Lớp: NT118.Q11.2

Điểm tự đánh giá
10

THÀNH VIÊN THỰC HIỆN (Nhóm xx):

STT	Họ và tên	MSSV
1	Nguyễn Lê Nhật Đăng	23520231

ĐÁNH GIÁ KHÁC:

Tổng thời gian thực hiện	2 ngày
Phân chia công việc	
Ý kiến *(nếu có)* + Khó khăn + Đề xuất, kiến nghị	Link chi tiết mã nguồn bài Lab: MobileApplicationDevelopment/LAB03 at main · tofumapu/MobileApplicationDevelopment

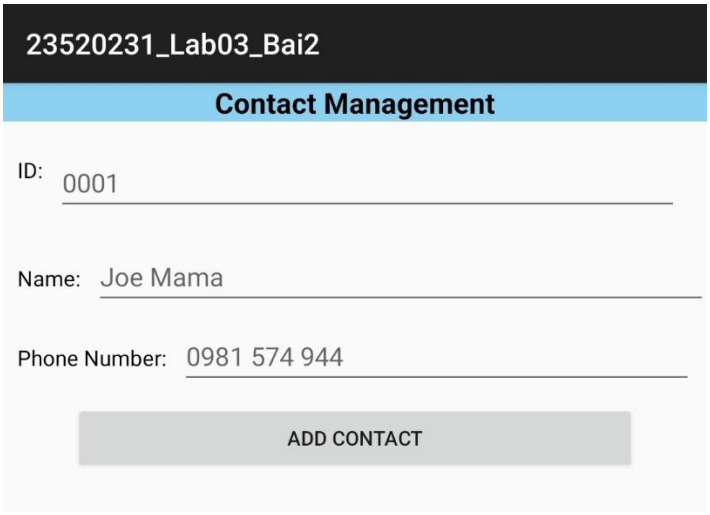
Phần bên dưới của báo cáo này là báo cáo chi tiết của cá nhân thực hiện

Mục lục

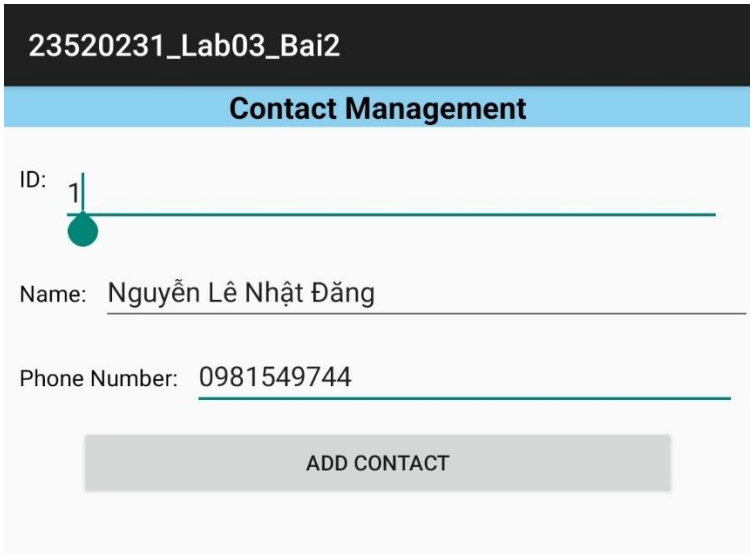
A. Bài thực hành quản lý liên lạc

0. Tổng quan

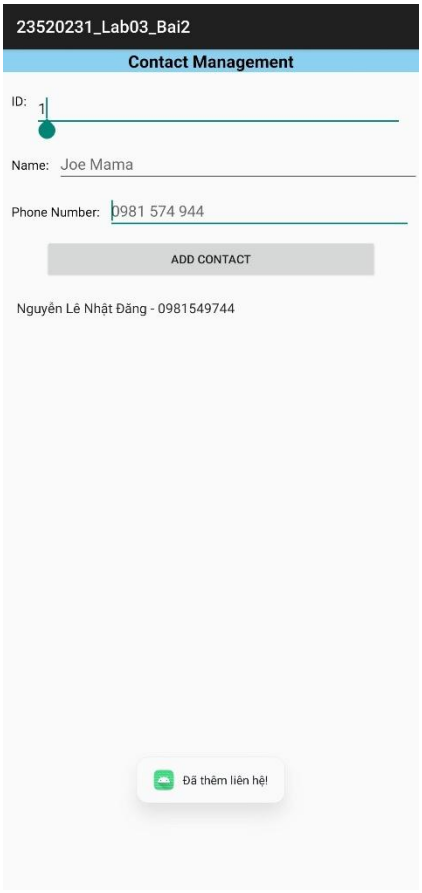
- Xây dựng giao diện quản lý liên lạc đơn giản, sử dụng ListView để quản lý danh sách liên lạc.



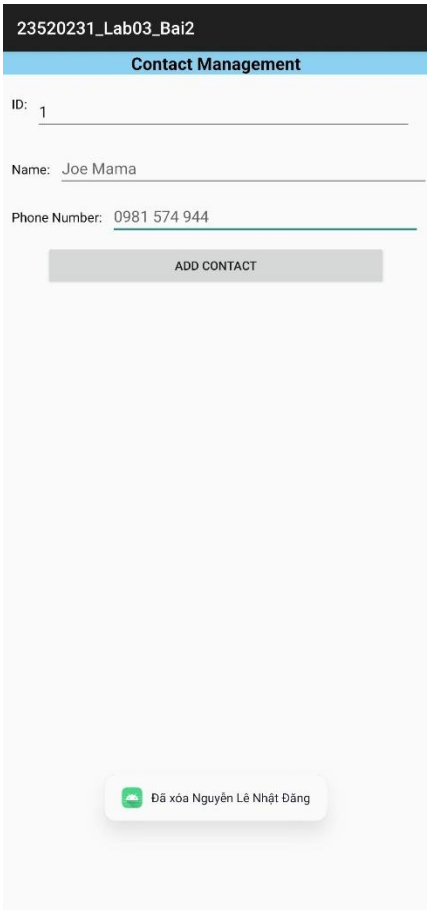
Hình 1: Giao diện quản lý liên lạc



Hình 2: Nhập thông tin liên lạc và nhấn button thêm liên lạc



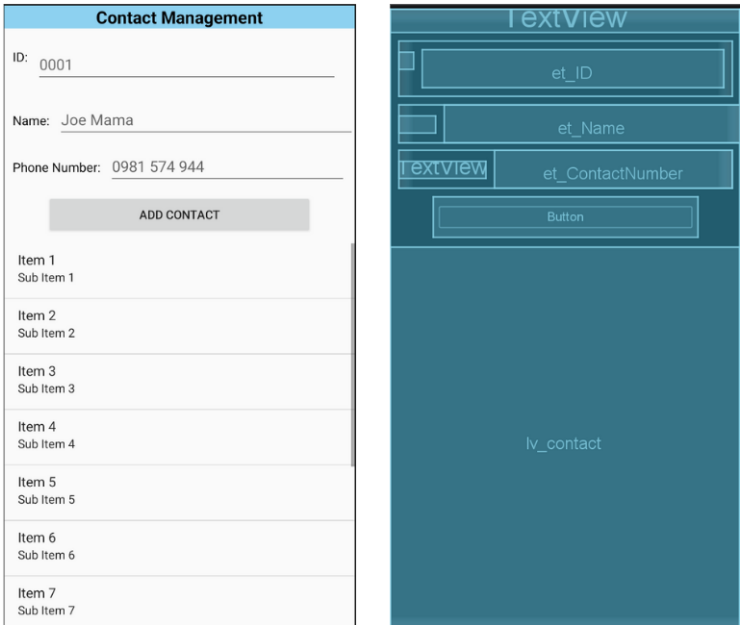
Hình 3: Liên lạc được thêm vào ListView thành công, make toast hiển thị thông báo thêm liên hệ thành công



Hình 4: Nhấn giữ vào liên lạc đó, sẽ lập tức xoá liên lạc và make toast hiển thị thông báo đã xoá liên lạc

1. Giao diện xml

- Xây dựng giao diện cơ bản với các TextView, EditText, Button và ListView cơ bản để quản lý thông tin liên lạc (phát triển từ các bài tập đã làm ở LAB2).

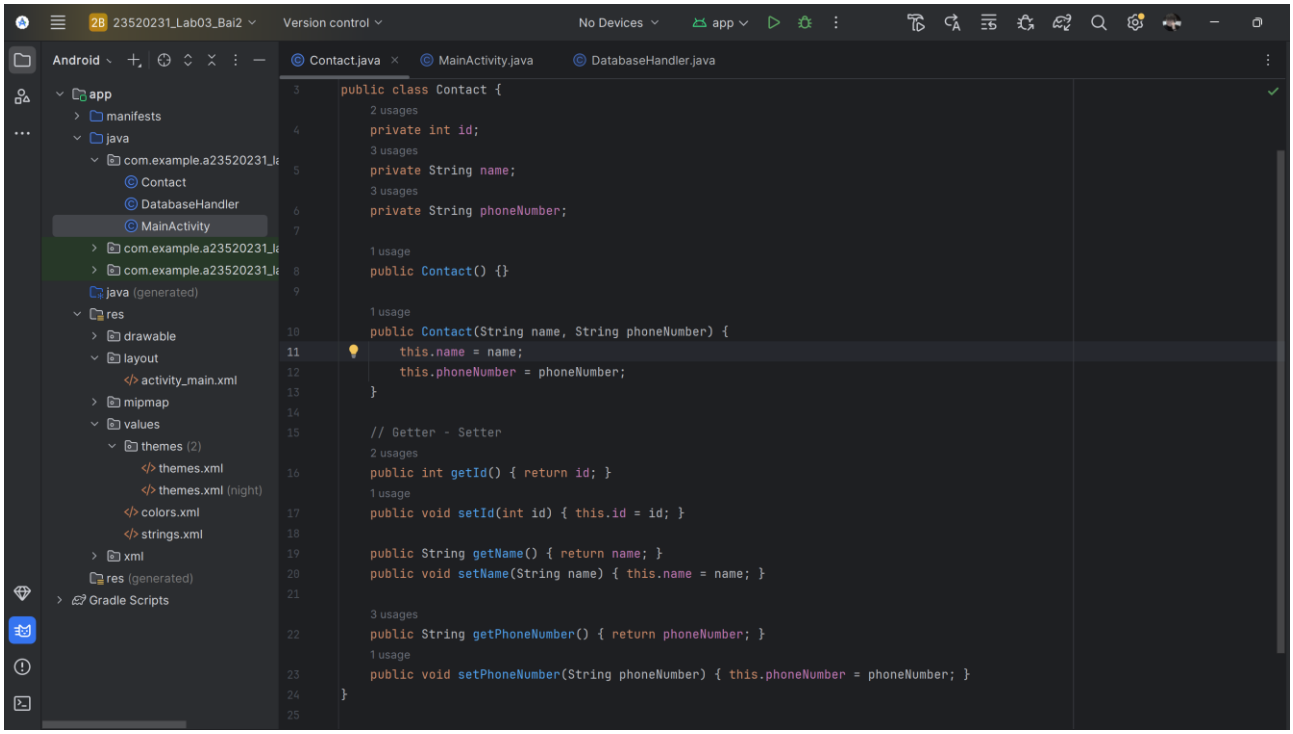


Hình 5: Giao diện quản lý liên lạc activity.xml

2. Các file .java

a. Contact.java

- Xây dựng class Contact với các trường như id thông tin liên lạc, tên liên lạc và số điện thoại liên lạc. Các hàm khởi tạo và getter/setter cơ bản.



Hình 6: Contact.java

b. DatabaseHandler.java

- Xây dựng lớp DatabaseHandler kế thừa SQLiteOpenHelper, là lớp trung gian giúp tạo, nâng cấp, và truy cập cơ sở dữ liệu SQLite trong Android.
- Hàm onCreate(): Tự động được gọi khi database được tạo lần đầu tiên.

```
@Override
public void onCreate(SQLiteDatabase db) {
    String CREATE_CONTACTS_TABLE = "CREATE TABLE " + TABLE_CONTACTS + "("
            + KEY_ID + " INTEGER PRIMARY KEY,"
            + KEY_NAME + " TEXT,"
            + KEY_PH_NO + " TEXT" + ")";
    db.execSQL(CREATE_CONTACTS_TABLE);
}
```

Hình 7: Hàm onCreate()

- Hàm onUpgrade(): Hàm được gọi khi phiên bản database tăng, hàm sẽ xoá bảng cũ và tạo lại bảng mới.

```
no usages
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    db.execSQL("DROP TABLE IF EXISTS " + TABLE_CONTACTS);
    onCreate(db);
}
```

Hình 8: Hàm onUpgrade()

- Hàm addContact(): Hàm thực hiện chức năng thêm một liên hệ mới vào bảng contacts, với content values là cấu trúc để lưu cột, giá trị. Sau khi được thêm các giá trị, tiến hành thêm vào bảng liên lạc. Cuối cùng, đóng kết nối database để tránh rò rỉ bộ nhớ.

```
1 usage
public void addContact(Contact contact) {
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues values = new ContentValues();
    values.put(KEY_NAME, contact.getName());
    values.put(KEY_PH_NO, contact.getPhoneNumber());
    db.insert(TABLE_CONTACTS, nullColumnHack: null, values);
    db.close();
}
```

Hình 9: Hàm addContact()

- Hàm getAllContacts(): Thực hiện truy vấn Select * from contacts. Sử dụng Cursor để duyệt qua từng dòng dữ liệu. Với mỗi dòng, lấy dữ liệu ở các cột theo chỉ mục. Tạo đối tượng Contact tương ứng. Và cuối cùng, thêm vào List<Contact>, trả về List là tất cả các liên lạc trong database.

```
1 usage
public List<Contact> getAllContacts() {
    List<Contact> contactList = new ArrayList<>();
    String selectQuery = "SELECT * FROM " + TABLE_CONTACTS;
    SQLiteDatabase db = this.getReadableDatabase();
    Cursor cursor = db.rawQuery(selectQuery, selectionArgs: null);

    if (cursor.moveToFirst()) {
        do {
            Contact contact = new Contact();
            contact.setId(cursor.getInt( i: 0));
            contact.setName(cursor.getString( i: 1));
            contact.setPhoneNumber(cursor.getString( i: 2));
            contactList.add(contact);
        } while (cursor.moveToNext());
    }
    cursor.close();
    db.close();
    return contactList;
}
```

Hình 10: Hàm getAllContacts()

- Hàm deteleContact(): thực hiện xoá đi liên hệ có id được truyền vào.

```
1 usage
public void deleteContact(int id) {
    SQLiteDatabase db = this.getWritableDatabase();
    db.delete(TABLE_CONTACTS, whereClause: KEY_ID + "=?", new String[]{String.valueOf(id)});
    db.close();
}
```

Hình 11: Hàm deleteContact()

c. MainAcivity.java

- Hàm chính có chức năng khởi tạo và ánh xạ các View ở activity_main.xml, xử lí sự kiện thêm liên lạc bằng Button Add Contact, khi giữ liên hệ, sẽ xoá liên lạc đi khỏi danh sách và cập nhật trên ListView.
- Hàm loadContacts(): Gọi dbHandler.getAllContacts() để lấy các Contact từ SQLite, sau đó duyệt từng contact và tạo danh sách chuỗi các tên – số điện thoại. Tạo array_adapter với layout list đơn giản. Cuối cùng gắn adapter cho ListView để hiển thị danh sách.

```
3 usages
private void loadContacts() {
    contactList = dbHandler.getAllContacts();
    contactNames = new ArrayList<>();
    for (Contact c : contactList) {
        contactNames.add(c.getName() + " - " + c.getPhoneNumber());
    }
    adapter = new ArrayAdapter<>( context: this, android.R.layout.simple_list_item_1, contactNames);
    lvContact.setAdapter(adapter);
}
```

Hình 12: Hàm loadContacts()

- Hàm xử lý nút thêm liên hệ: Khi người dùng thêm liên hệ, sẽ lấy giá trị từ các EditText tên và số điện thoại thêm vào database. Thực hiện loadContacts() để cập nhật danh sách trên ListView. Cuối cùng hiển thị thông báo lên màn hình là đã thêm liên hệ thành công hoặc hiển thị cảnh báo khi người dùng không nhập đủ thông tin.

```
btnAdd.setOnClickListener( View v -> {
    String name = etName.getText().toString();
    String phone = etPhone.getText().toString();
    if (!name.isEmpty() && !phone.isEmpty()) {
        dbHandler.addContact(new Contact(name, phone));
        loadContacts();
        etName.setText("");
        etPhone.setText("");
        Toast.makeText( context: this,  text: "Đã thêm liên hệ!", Toast.LENGTH_SHORT).show();
    } else {
        Toast.makeText( context: this,  text: "Vui lòng nhập đủ thông tin!", Toast.LENGTH_SHORT).show();
    }
});
```

Hình 13: Xử lý thêm một liên hệ

- Hàm xử lý khi người dùng nhấn giữ vào một liên hệ: Khi người dùng nhấn giữ lâu vào một đối tượng trong ListView, lấy đối tượng Contact tương ứng, gọi hàm deleteContact để xoá đối tượng trong database, cuối cùng gọi hàm loadContacts() để cập nhật lại danh sách liên hệ trên ListView. Cuối cùng, hiển thị thông báo đã xoá liên hệ đó liên màn hình thông báo cho người dùng.

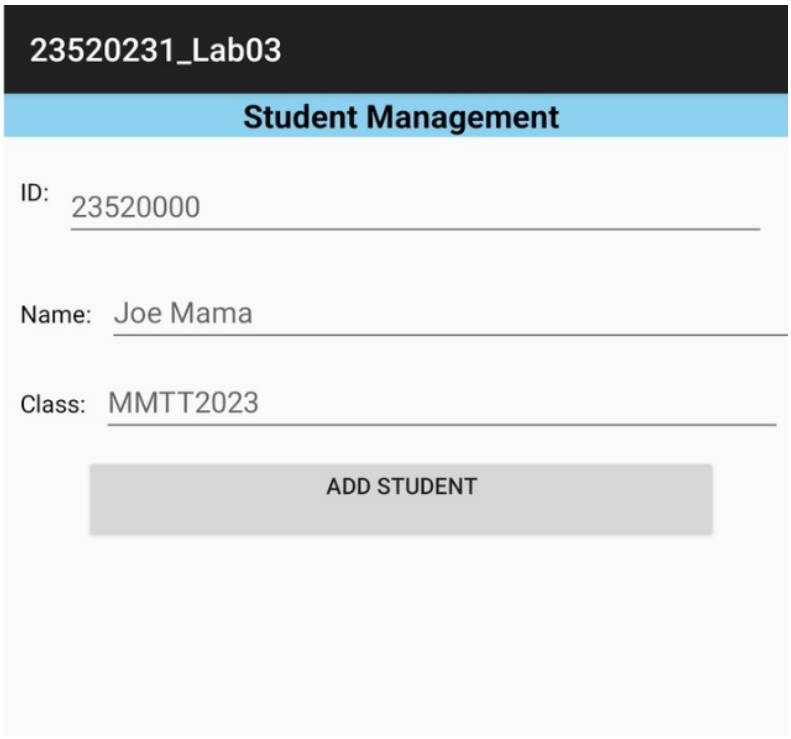
```
// Xóa contact bằng long click
lvContact.setOnItemLongClickListener(( AdapterView<?> parent,  View view,  int position,  long id) -> {
    Contact contact = contactList.get(position);
    dbHandler.deleteContact(contact.getId());
    loadContacts();
    Toast.makeText( context: this,  text: "Đã xóa " + contact.getName(), Toast.LENGTH_SHORT).show();
    return true;
});
```

Hình 14: Xoá một liên hệ bằng thao tác long click

B. Bài thực hành quản lý sinh viên

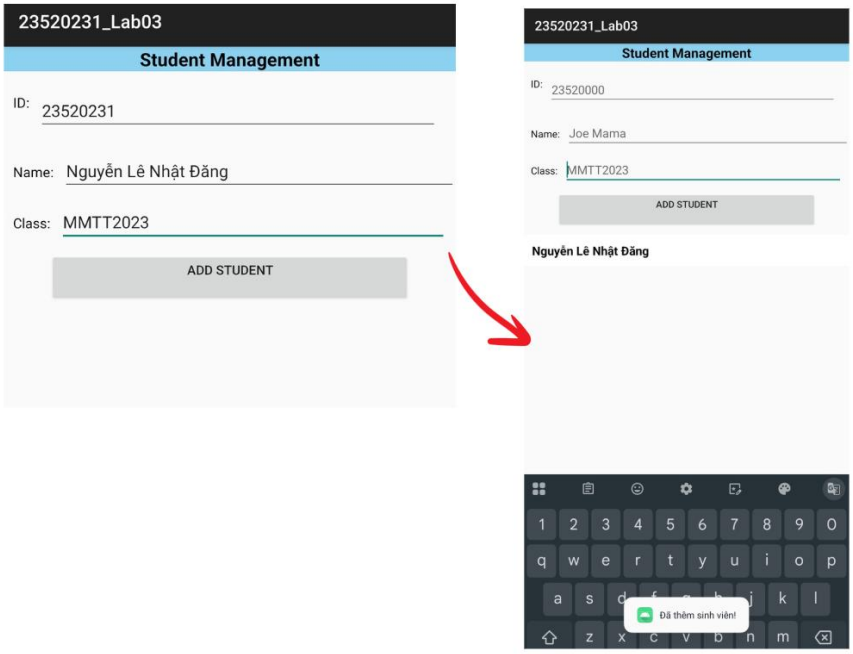
0. Tổng quan

- Xây dựng ứng dụng quản lý sinh viên với RecyclerView và SQLite. Có các chức năng thêm, xoá, sửa, nhấn để xem chi tiết sinh viên đó, nhấn giữ để xoá đi sinh viên khỏi danh sách.



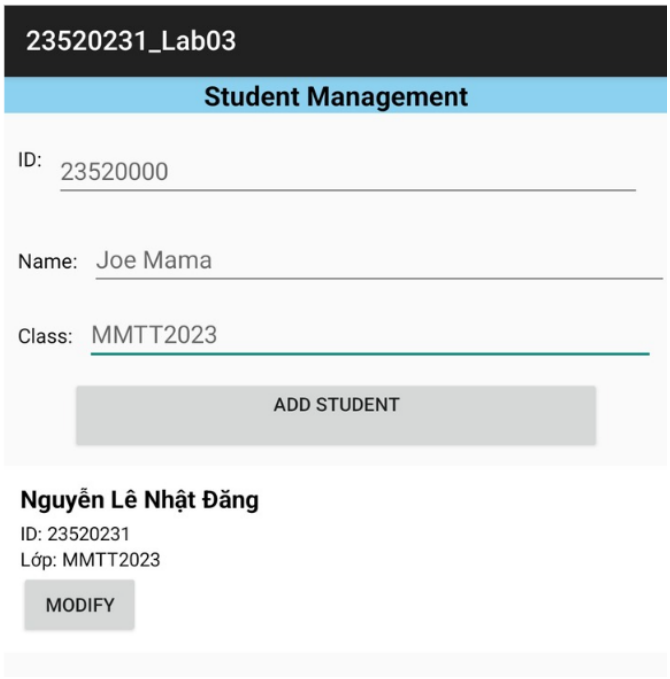
Hình 15: Kết quả tổng quan giao diện khi khởi chạy

- Tiến hành thêm một sinh viên vào và nhấn nút thêm sinh viên:



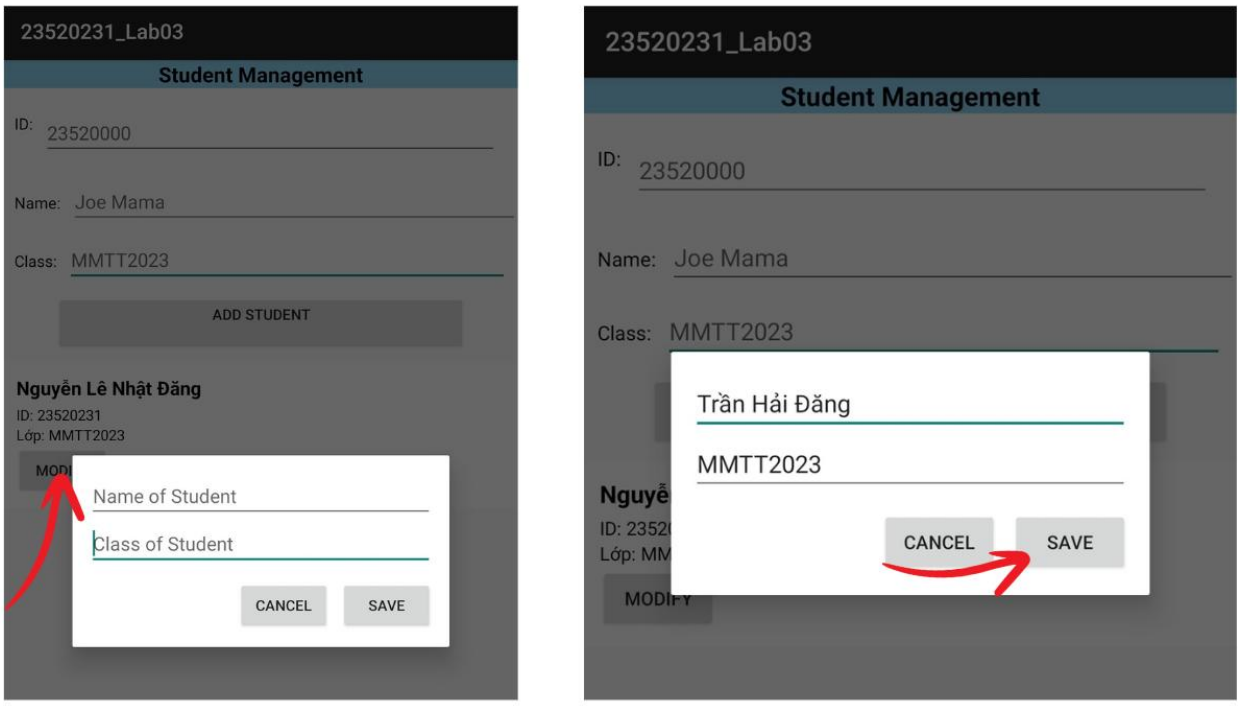
Hình 16: Kết quả khi thêm mới một sinh viên, sinh viên mới sẽ được thêm vào RecyclerView và thông báo Toast thông báo đã thêm thành công

- Khi nhấn vào item sinh viên đó trong RecyclerView, sẽ hiển thị rõ hơn chi tiết (ID, Lớp đang học) sinh viên đó.



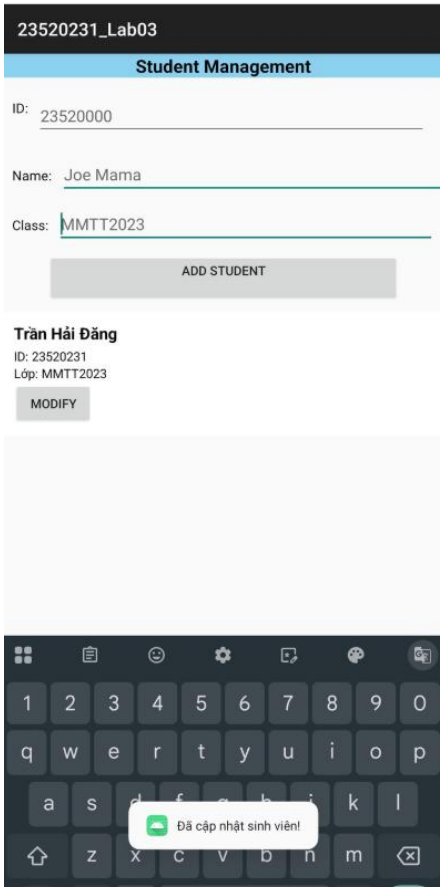
Hình 17: Kết quả khi nhấn 1 lần vào item trong RecyclerView

- Khi nhấn 1 lần vào item, nếu người dùng muốn sửa đổi thông tin, nhấn vào "Modify". Khi đó, một giao diện mới hiện ra cho phép người dùng sửa đổi mọi thông tin (trừ ID của sinh viên vì đây là khoá chính). Nhấn "Save" để lưu thông tin sửa đổi hoặc "Cancel" để huỷ bỏ sửa đổi.



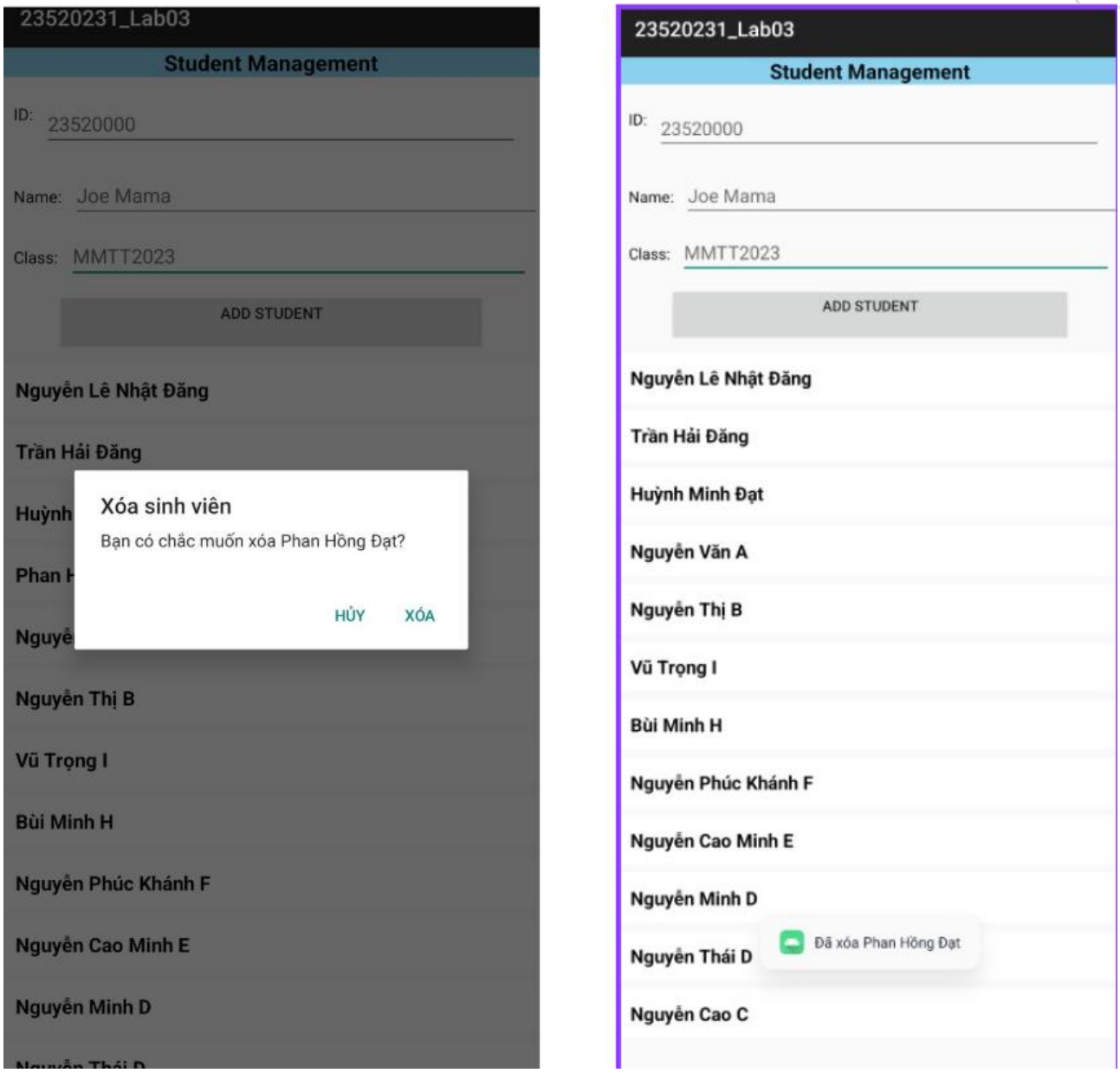
Hình 18: Thay đổi thông tin sinh viên trong giao diện chỉnh sửa

- Khi ấn "Save", dữ liệu các trường sẽ được "Update" trong database và thay đổi thông tin trong RecyclerView và thông báo Toast ra màn hình đã cập nhật sinh viên.



Hình 19: Kết quả khi thay đổi thông tin sinh viên

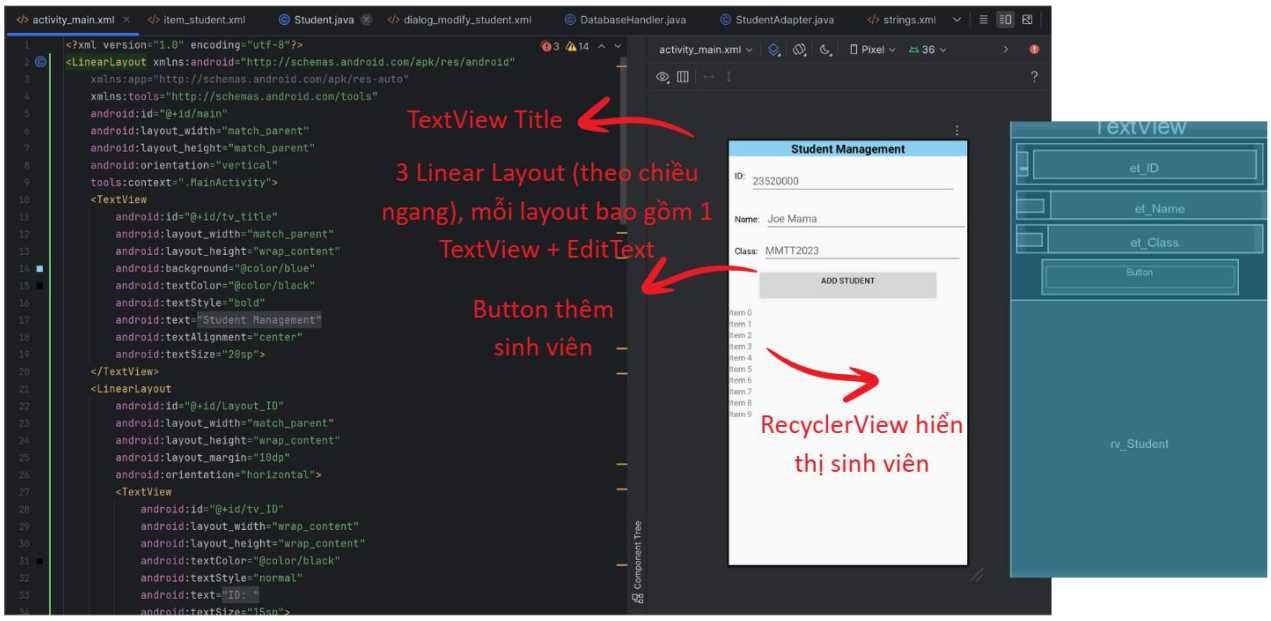
- Khi giao diện có nhiều sinh viên, RecyclerView có thể tuỳ biến cuộn lên/xuống để xem thông tin sinh viên trong LinearLayout.
- Kết quả thử xoá một sinh viên bằng cách nhấn giữ vào Item đó trong RecyclerView.



Hình 20: Kết quả khi xoá một sinh viên trong RecyclerView bằng cách nhấn giữ

1. Giao diện .xml

a. activity_main.xml

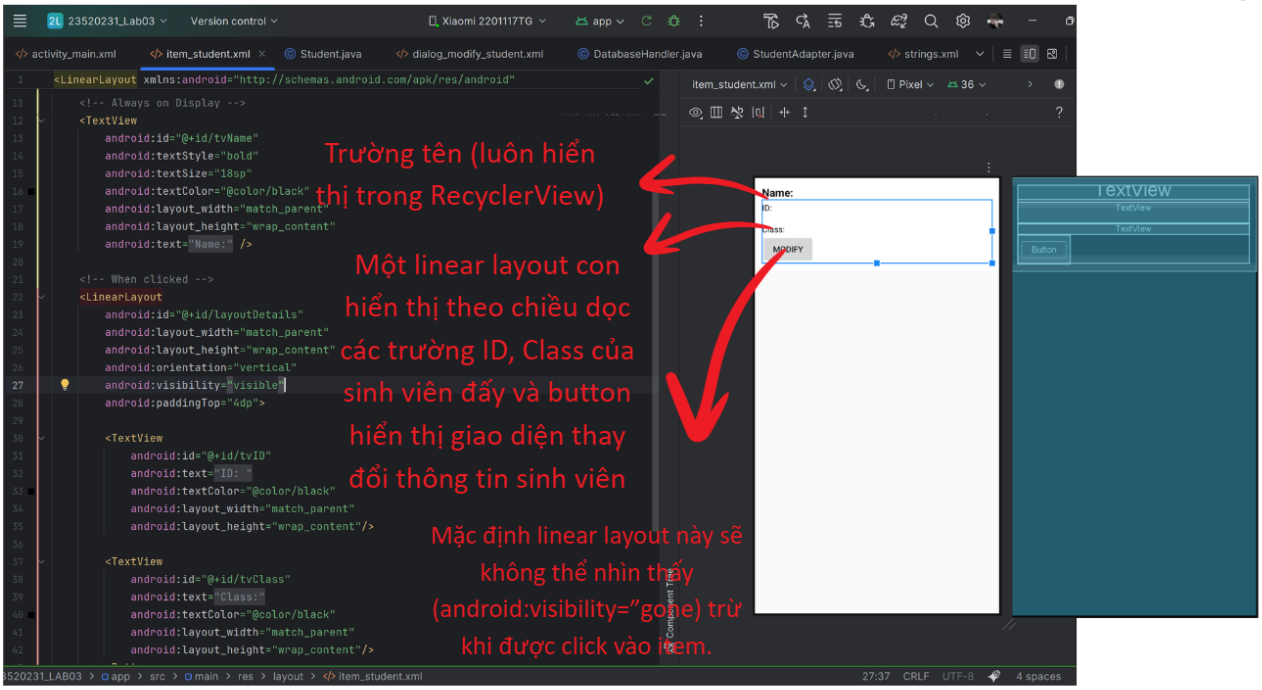


Hình 21: Thiết kế của activity_main.xml

- Bố cục của activity_main.xml xây dựng bằng LinearLayout (theo chiều dọc) gồm:
    - TextView title: Tiêu đề hiển thị ở phía trên cùng Layout cha.
    - 3 LinearLayout con (theo chiều ngang): mỗi Layout bao gồm một TextView và EditText hiển thị (ID sinh viên, Tên sinh viên, Lớp của sinh viên).
    - Button thêm sinh viên: Button có chức năng thêm một item (sinh viên) với các trường (ID, Name, Class) vào RecyclerView.
    - RecyclerView: Hiển thị các item (sinh viên) ra màn hình giao diện.

b. Item_student.xml

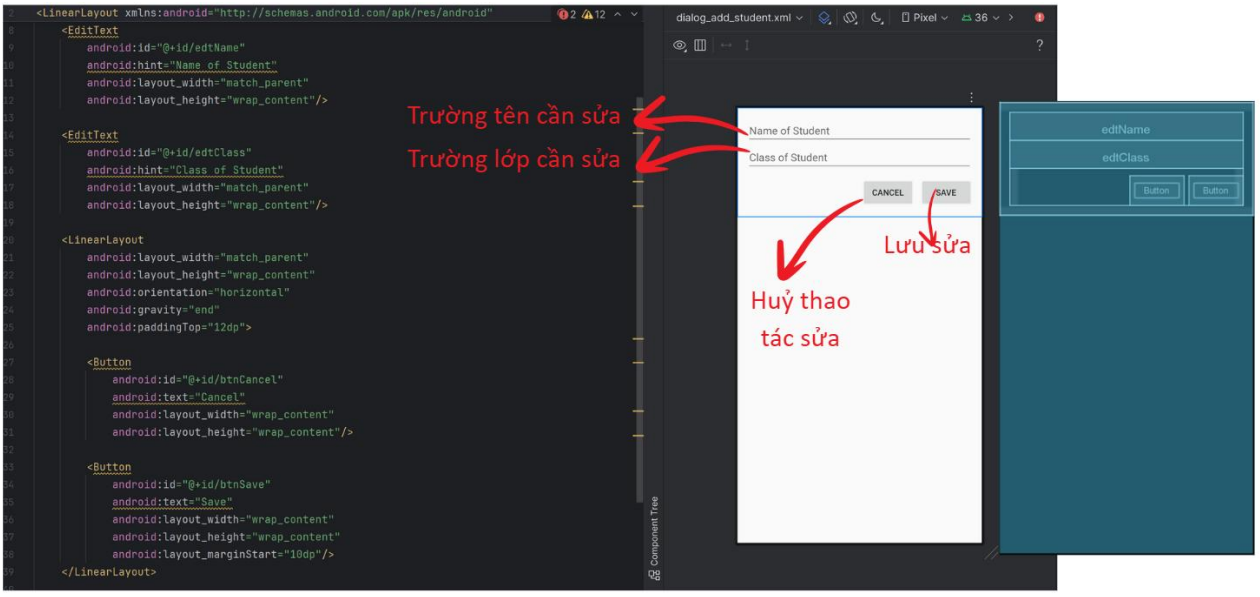
- Giao diện là một LinearLayout (thiết kế theo chiều ngang) để thiết kế cách mà một item hiển thị trong RecyclerView. Bao gồm:
    - Trường tên (trường luôn hiển thị trong RecyclerView).
    - Một linear layout con hiển thị theo chiều dọc các trường ID, Class của sinh viên đấy và button hiển thị giao diện thay đổi thông tin sinh viên (Mặc định linear layout này sẽ không thể nhìn thấy (android:visibility=”gone) trừ khi được click vào item.).



Hình 22: Thiết kế item_student.xml

c. dialog_modify_student.xml

- Giao diện được thiết kế bằng LinearLayout (theo chiều dọc), dùng để chỉnh sửa thông tin item (sinh viên) khi nhấn vào nút Modify. Bao gồm:
  - EditText nhập tên sinh viên cần sửa.
  - EditText nhập lớp sinh viên cần sửa.
  - Nút cancel để huỷ thao tác sửa.
  - Nút save để lưu thông tin đã sửa. Cập nhật database và RecyclerView, hiển thị Toast báo cập nhật thành công lên màn hình.



Hình 23: Thiết kế giao diện dialog_modify_student.xml

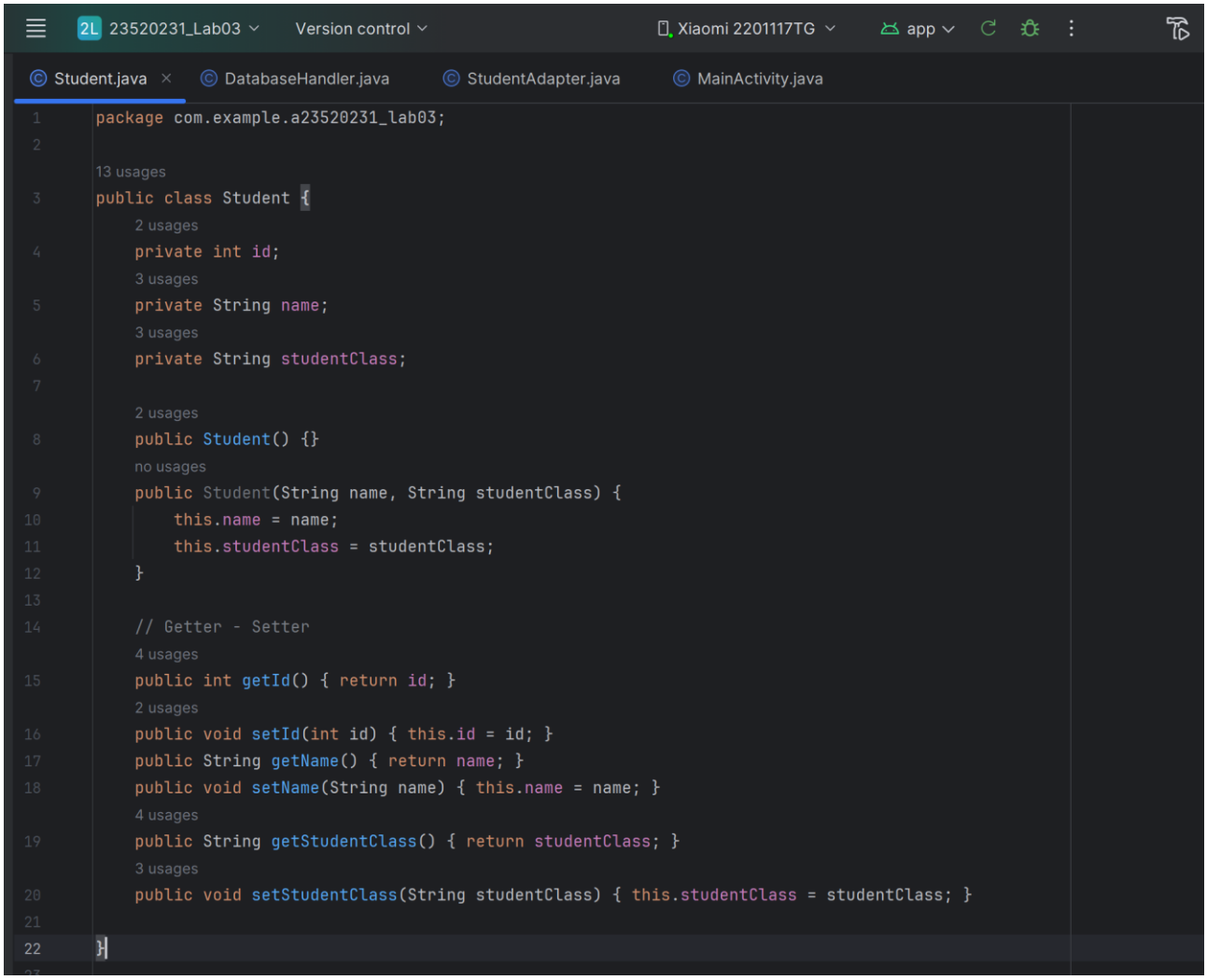
2. Các file .java.

a. Cấu trúc Database Student

Columns	DataType	Key
id	INTEGER	Primary key (không tự tăng, người dùng tự nhập).
name	TEXT	
studentClass	TEXT	

b. Lớp Student (Student.java)

- Khởi tạo lớp Student chứa các biến (ID, Name và Class) và các constructor, getter/setter cần thiết.



Hình 24: Student.java

c. DatabaseHandler.java

- Hàm onCreate(): Hàm được gọi khi chạy app lần đầu tiên, hàm có chức năng tạo bảng Student bằng lệnh SQL “CREATE TABLE”.

```
@Override
public void onCreate(SQLiteDatabase db) {
    String CREATE_TABLE = "CREATE TABLE " + TABLE_STUDENTS + "("
            + KEY_ID + " INTEGER PRIMARY KEY,"
            + KEY_NAME + " TEXT,"
            + KEY_CLASS + " TEXT)";
    db.execSQL(CREATE_TABLE);
}
```

Hình 25: Hàm onCreate()

- Hàm onUpgrade(): có chức năng xoá toàn bộ dữ liệu mới, tạo lại bảng mới từ đầu.

```
no usages
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    db.execSQL("DROP TABLE IF EXISTS " + TABLE_STUDENTS);
    onCreate(db);
}
```

Hình 26: Hàm onUpgrade()

- Hàm addStudent(): Mở database bằng lệnh getWriteableDatabase(), tạo các content values để chứa dữ liệu, sau đó insert vào table bằng lệnh db.insert. Cuối cùng, đóng lại database.

```
1 usage
public void addStudent(Student student) {
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues values = new ContentValues();
    values.put(KEY_ID, student.getId());
    values.put(KEY_NAME, student.getName());
    values.put(KEY_CLASS, student.getStudentClass());
    db.insert(TABLE_STUDENTS,  nullColumnHack: null, values);
    db.close();
}
```

Hình 27: Hàm addStudent()

- Hàm getAllStudents(): Chức năng là đọc toàn bộ sinh viên, sử dụng cursor để đọc nội dung trong bảng student, duyệt cursor để đọc từng dòng, đưa các đối tượng Student vào List<Student>. Cuối cùng đóng cursor và database, kết quả là trả về danh sách sinh viên hiển thị trong RecyclerView.

```
public List<Student> getAllStudents() {
    List<Student> list = new ArrayList<>();
    SQLiteDatabase db = this.getReadableDatabase();
    Cursor cursor = db.rawQuery( sql: "SELECT * FROM " + TABLE_STUDENTS, selectionArgs: null);
    if (cursor.moveToFirst()) {
        do {
            Student st = new Student();
            st.setId(cursor.getInt( i: 0));
            st.setName(cursor.getString( i: 1));
            st.setStudentClass(cursor.getString( i: 2));
            list.add(st);
        } while (cursor.moveToNext());
    }
    cursor.close();
    db.close();
    return list;
}
```

Hình 28: Hàm getAllStudents()

- Hàm updateStudent(): Chức năng sửa thông tin sinh viên. Mở database bằng getWritableDatabase(). Set lại các giá trị Name và Class. Sau đó update lại thông tin bằng cách cập nhật theo id khớp với id cần modify.

```
1 usage
public int updateStudent(Student student) {
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues values = new ContentValues();
    values.put(KEY_NAME, student.getName());
    values.put(KEY_CLASS, student.getStudentClass());
    return db.update(TABLE_STUDENTS, values, whereClause: KEY_ID + "=?", new String[]{String.valueOf(student.getId())});
}
```

Hình 29: Hàm updateStudent()

- Hàm deleteStudent(): Chức năng xoá sinh viên theo ID. Sau đó sẽ đóng database.

```
public void deleteStudent(int id) {
    SQLiteDatabase db = this.getWritableDatabase();
    db.delete(TABLE_STUDENTS, whereClause: KEY_ID + "=?", new String[]{String.valueOf(id)});
    db.close();
}
```

Hình 30: Hàm deleteStudent()

d. StudentAdapter.java

- Khai báo thuộc tính lớp StudentAdapter và tạo Constructor: Khai báo danh sách sinh viên, context (cho phép Adapter gọi Toast, AlertDialog và Inflate Layout). Các thao tác với SQLite (thêm/đọc/cập nhật/xoá).

```
1 usage
public StudentAdapter(Context context, List<Student> list, DatabaseHandler dbHandler) {
    this.context = context;
    this.studentList = list;
    this.dbHandler = dbHandler;
}
```

Hình 31: Khai báo Constructor lớp StudentAdapter

- onCreateViewHolder(): Inflate layout (layout hiển thị một sinh viên) và trả về một StudentViewHolder chứa toàn bộ view con. Mỗi item trong RecyclerView tương ứng là một Student.

```
@NonNull
@Override
public StudentViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
    View view = LayoutInflater.from(context).inflate(R.layout.item_student, parent, attachToRoot: false);
    return new StudentViewHolder(view);
}
```

Hình 32: onCreateViewHolder()

- onBindViewHolder(): Chức năng gán dữ liệu và xử lý sự kiện. Với sự kiện click, phần Layout chi tiết sẽ được hiện ra (trường Name và Class) và ngược lại, khi click lại sẽ đóng lại phần layout hiển thị thông tin chi tiết. Khi click vào nút modify, sẽ hiển thị thông tin cũ cho người dùng sửa và cập nhật giá trị lên database và RecyclerView.

```
public class StudentAdapter extends RecyclerView.Adapter<StudentAdapter.StudentViewHolder> {

    @Override
    public void onBindViewHolder(@NonNull StudentViewHolder holder, int position) {
        Student st = studentList.get(position);

        holder.tvName.setText(st.getName());
        holder.tvID.setText("ID: " + st.getId());
        holder.tvClass.setText("Lớp: " + st.getStudentClass());

        holder.itemView.setOnClickListener( View v -> {
            if (holder.layoutDetails.getVisibility() == View.GONE)
                holder.layoutDetails.setVisibility(View.VISIBLE);
            else
                holder.layoutDetails.setVisibility(View.GONE);
        });
        holder.btnModify.setOnClickListener( View v -> {
            // Inflate layout dialog
            View dialogView = LayoutInflater.from(context).inflate(R.layout.dialog_modify_student, root: null);
            EditText edtName = dialogView.findViewById(R.id.edtName);
            EditText edtClass = dialogView.findViewById(R.id.edtClass);
            Button btnCancel = dialogView.findViewById(R.id.btnCancel);
            Button btnSave = dialogView.findViewById(R.id.btnSave);

            // Gán giá trị hiện tại
            edtName.setText(st.getName());
            edtClass.setText(st.getStudentClass());

            AlertDialog dialog = new AlertDialog.Builder(context)
                    .setView(dialogView)
                    .create();
```

Hình 33: onBindViewHolder() (P1)

- Hiển thị toast thông báo đã cập nhật thành công và đóng dialog chỉnh sửa thông tin.

```
        btnCancel.setOnClickListener( View cancelView -> dialog.dismiss());

        btnSave.setOnClickListener( View saveView -> {
            String newName = edtName.getText().toString().trim();
            String newClass = edtClass.getText().toString().trim();

            if (newName.isEmpty() || newClass.isEmpty()) {
                Toast.makeText(context,  text: "Vui lòng nhập đầy đủ thông tin!", Toast.LENGTH_SHORT).show();
                return;
            }

            // Cập nhật đối tượng trong database
            st.setName(newName);
            st.setStudentClass(newClass);
            dbHandler.updateStudent(st);

            // Cập nhật lại giao diện
            notifyItemChanged(holder.getBindingAdapterPosition());
            Toast.makeText(context,  text: "Đã cập nhật sinh viên!", Toast.LENGTH_SHORT).show();

            dialog.dismiss();
        });

        dialog.show();
    });
```

Hình 34: onBindViewHolder() (P2)

- Khi nhấn giữ vào một item trong RecyclerView, sẽ hiển thị AlertDialog thông báo người dùng muốn xoá sinh viên hay không? Nếu không, sẽ tiến hành huỷ việc xoá và ngược lại, sẽ xoá ra khỏi SQLite, xoá ra khỏi StudentList hiện tại và cập nhật lại trên RecyclerView.

```
    holder.itemView.setOnLongClickListener( View v -> {
        new AlertDialog.Builder(context)
                .setTitle("Xóa sinh viên")
                .setMessage("Bạn có chắc muốn xóa " + st.getName() + "?")
                .setPositiveButton( text: "Xóa", ( DialogInterface dialog,  int which) -> {
                    int currentPosition = holder.getAdapterPosition();
                    if (currentPosition != RecyclerView.NO_POSITION && currentPosition < studentList.size()) {
                        dbHandler.deleteStudent(st.getId());
                        studentList.remove(currentPosition);
                        notifyItemRemoved(currentPosition);
                        notifyItemRangeChanged(currentPosition, studentList.size());
                        Toast.makeText(context,  text: "Đã xóa " + st.getName(), Toast.LENGTH_SHORT).show();
                    }
                })
                .setNegativeButton( text: "Hủy",  listener: null)
                .show();
        return true;
    });

}
```

Hình 35: onBindViewHolder() (P3)

- Hàm getItemCount() cho biết số lượng phần tử cần hiển thị trên RecyclerView và khai báo các View + Ánh xạ.

```
@Override
public int getItemCount() {
    return studentList.size();
}

4 usages
static class StudentViewHolder extends RecyclerView.ViewHolder {
    2 usages
    TextView tvName, tvID, tvClass;
    4 usages
    LinearLayout layoutDetails;
    2 usages
    Button btnModify;

    1 usage
    public StudentViewHolder(@NonNull View itemView) {
        super(itemView);
        tvName = itemView.findViewById(R.id.tvName);
        tvID = itemView.findViewById(R.id.tvID);
        tvClass = itemView.findViewById(R.id.tvClass);
        layoutDetails = itemView.findViewById(R.id.layoutDetails);
        btnModify = itemView.findViewById(R.id.btn_Modify);

    }
}
}
```

Hình 36: getItemCount() và khởi tạo, ánh xạ View

e. MainActivity.java

- Đầu tiên, khởi tạo các biến cần thiết và ánh xạ chúng.
- Khởi tạo dbHandler đã khai báo ở bước c) giúp thực hiện các câu lệnh SQL. Sau đó, đọc bảng dữ liệu thông qua hàm getAllStudents(), hiển thị danh sách LinearLayout theo chiều dọc và gắn Adapter này vào RecyclerView.
- Khi nhấn vào button thêm sinh viên, sẽ gọi hàm thêm sinh viên addStudent().

```
    // Nạp dữ liệu
    studentList = dbHandler.getAllStudents();
    adapter = new StudentAdapter( context: this, studentList, dbHandler);

    rvStudents.setLayoutManager(new LinearLayoutManager( context: this));
    rvStudents.setAdapter(adapter);

    btnAdd.setOnClickListener( View v -> addStudent());
```

Hình 37: Nạp dữ liệu tại MainActivity.java

- Tại addStudent(), tiến hành kiểm tra các trường, nếu các trường chưa được nhập -> hiển thị cảnh báo người dùng nhập. Khi đã đầy đủ thông tin, tạo đối tượng Student (bao gồm các trường id, name và class). Tiếp theo, thêm đối tượng Student vào database bằng cách gọi đến hàm addStudent() trong DatabaseHandler và thực hiện lệnh SQL. Cuối cùng, cập nhật lại RecyclerView (bằng cách xoá hết StudentList và cập nhật lại từ đầu) và tạo Toast hiển thị thông báo đã thêm sinh viên thành công.

```
private void addStudent() {
    String idText = etID.getText().toString().trim();
    if (idText.isEmpty()) {
        Toast.makeText( context: this,  text: "Vui lòng nhập ID!", Toast.LENGTH_SHORT).show();
        return;
    }
    int id = Integer.parseInt(idText);
    String name = etName.getText().toString().trim();
    String cls = etClass.getText().toString().trim();

    if (name.isEmpty() || cls.isEmpty()) {
        Toast.makeText( context: this,  text: "Vui lòng nhập đủ thông tin!", Toast.LENGTH_SHORT).show();
        return;
    }

    Student s = new Student();
    s.setId(id);
    s.setName(name);
    s.setStudentClass(cls);
    dbHandler.addStudent(s);

    // Cập nhật lại danh sách
    studentList.clear();
    studentList.addAll(dbHandler.getAllStudents());
    adapter.notifyDataSetChanged();

    // Xóa text input
    etID.setText("");
    etName.setText("");
    etClass.setText("");

    Toast.makeText( context: this,  text: "Đã thêm sinh viên!", Toast.LENGTH_SHORT).show();
}
```

Hình 38: Hàm addStudent()