Contact Management App

Simplified Contact Handling

A user-friendly app for adding, editing, viewing, and deleting contacts, designed to streamline contact management with intuitive features and seamless navigation.

```
class DatabaseHelper(context: Context) :SQLiteOpenHelper(context,DATABASE_NAME, factory: null,DATABASE_VERSION){
    companion object{
        private const val DATABASE_NAME = "ContactDatabase"
        private const val DATABASE_VERSION = 1

        val TABLE_NAME = "contacts"
        val COLUMN_ID = "id"
        val COLUMN_NAME = "name"
        val COLUMN_PHONE = "phone"
        val COLUMN_IMAGE = "image"
    }

    Edit | Explain | Test | Document | Fix
    override fun onCreate(db: SQLiteDatabase?) {
        val createTableQuery = "CREATE TABLE $TABLE_NAME ($COLUMN_ID INTEGER PRIMARY KEY, $COLUMN_NAME TEXT, $COLUMN_PHONE TEXT, $COLUMN_IMAGE TEXT)"
        db?.execSQL(createTableQuery)
    }

    Edit | Explain | Test | Document | Fix
    override fun onUpgrade(db: SQLiteDatabase?, oldVersion: Int, newVersion: Int) {
        db?.execSQL( sql: "DROP TABLE IF EXISTS $TABLE_NAME")
        onCreate(db)
    }
```

Create Database Helper Class

- Manages database creation, upgrades, and queries (CRUD operations).
- Encapsulates complex SQL logic, keeping your app code clean.
- Promotes reuse and reduces duplication.

Create Data Class

Purpose: Represents structured data in a clean and concise way.

- DatabaseHelper handles database logic, while Data Class models the data.
- Promotes separation of concerns, making code modular and maintainable.
- Example: Fetch a user from the database and return it as a User object.

```
package com.pixeleye.contact

Edit | Explain | Test | Document | Fix
data class Contact(
    val id: Int,
    val name: String,
    val phone: String,
    val imagePath: String
)
```

InsertContactActivity

```
18  class InsertContactActivity : AppCompatActivity() {
19      private lateinit var dbHelper: DatabaseHelper
20      private var selectedImagePath: String = ""
        Edit | Explain | Test | Document | Fix
21      override fun onCreate(savedInstanceState: Bundle?) {
22          super.onCreate(savedInstanceState)
23          setContentView(R.layout.activity_insert_contact)
24
25          dbHelper = DatabaseHelper( context: this)
26
27          val nameInput: EditText = findViewById(R.id.nameInput)
28          val phoneInput: EditText = findViewById(R.id.phoneInput)
29          val selectImageButton: Button = findViewById(R.id.selectImage)
30          val saveButton: Button = findViewById(R.id.saveButton)
31
```

Properties:

- **dbHelper**: Manages database operations.
- **selectedImagePath**: Stores the path of the selected contact image.

onCreate Method:

- Initializes the activity and links the layout (**activity_insert_contact**).
- Sets up **dbHelper** for database interactions.

UI Components:

- **nameInput**: Input field for the contact's name.
- **phoneInput**: Input field for the contact's phone number.
- **selectImageButton**: Button to choose a contact image.
- **saveButton**: Button to save contact details to the database.

Image Path Set Up

Select Image:

- When the "Select Image" button is clicked, an intent opens the device's gallery to choose an image.

Handle Image Selection:

- The onActivityResult method processes the selected image when the user picks one.

```
        // Select Image
        selectImageButton.setOnClickListener {
            val intent = Intent(Intent.ACTION_PICK)
            intent.type = "image/*"
            startActivityIfNeeded(intent, requestCode: 100)
        }
    }

    Edit | Explain | Test | Document | Fix
    override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
        super.onActivityResult(requestCode, resultCode, data)
        if (requestCode == 100 && resultCode == Activity.RESULT_OK) {
            val uri = data?.data
            uri?.let {
                // Show the selected image
                val profileImage: ImageView = findViewById(R.id.imageView)
                profileImage.setImageURI(uri)

                // Save the image to internal storage and get the file path
                val filePath = saveImageToFile( context: this, uri, fileName: "profile_image_${System.currentTimeMillis()}.png")

//                  Store the file path in the database
                if (filePath != null) {
                    selectedImagePath = filePath
                } else {
                    Toast.makeText( context: this, text: "Failed to save image", Toast.LENGTH_SHORT).show()
                }
            }
        }
    }
```

Preview the Image:

- The selected image is displayed in an ImageView for the user to see.

Save the Image:

- The image is saved to internal storage with a unique filename, and the file path is stored for later use.

Steps for

Saving a Contact

Capture User Input:

- Retrieve the name and phone number from input fields.

Validate Inputs:

- Check if both fields are filled before proceeding.

Create Contact Object:

- Create a Contact object with the input data and image path.

Save the Contact:

- Call the addContact method to save the contact in the database.

Display Success or Failure:

- Show a success message if the contact is saved, or an error message if it fails.

```
31
32      // Save Contact
33      saveButton.setOnClickListener {
34          val name = nameInput.text.toString()
35          val phone = phoneInput.text.toString()
36
37          if (name.isNotEmpty() && phone.isNotEmpty()) {
38              val contact = Contact(id = 0,name = name, phone = phone, imagePath = selectedImagePath)
39              val result = dbHelper.addContact(contact)
40              if (result > 0) {
41                  Toast.makeText( context: this,  text: "Contact saved successfully!", Toast.LENGTH_SHORT).show()
42                  finish()
43                  val intent = Intent( packageContext: this,MainActivity::class.java)
44                  startActivity(intent)
45              } else {
46                  Toast.makeText( context: this,  text: "Failed to save contact!", Toast.LENGTH_SHORT).show()
47              }
48          } else {
49              Toast.makeText( context: this,  text: "All fields are required!", Toast.LENGTH_SHORT).show()
50          }
51      }
52
```

```kotlin
class MainActivity : AppCompatActivity() {
    private lateinit var addButton: FloatingActionButton
    private lateinit var emptyText: TextView

    Edit | Explain | Test | Document | Fix
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        addButton = findViewById(R.id.addButton)
        emptyText = findViewById(R.id.emptyText)
        emptyText.visibility = View.GONE

        addButton.setOnClickListener {
            val intent = Intent( packageContext: this, InsertContactActivity::class.java)
            startActivity(intent)
        }
        setList()
    }

    Edit | Explain | Test | Document | Fix
    private fun setList() {
        val recyclerView: RecyclerView = findViewById(R.id.recyclerView)
        recyclerView.layoutManager = LinearLayoutManager( context: this)

        val dbHelper = DatabaseHelper( context: this)
        val contacts = dbHelper.getAllContacts()

        if (contacts.isEmpty()) {
            emptyText.visibility = View.VISIBLE
        }

        val adapter = ContactAdapter(contacts)
        recyclerView.adapter = adapter
    }
```

Main Activity

& Managing the Contact List

Main Activity (MainActivity)

- Entry point of the app, managing the main screen where contacts are displayed.

UI Elements

- Add Button (**addButton**): A floating action button to add new contacts.
- Empty Text (**emptyText**): Displays a message when no contacts are available.

onCreate Method

- Sets up the UI and initializes click behavior for the Add Button to navigate to **InsertContactActivity**.

setList Method

- Configures the **RecyclerView** with a **LinearLayoutManager**.
- Fetches all contacts from the database and displays them using the **ContactAdapter**.
- Shows the **emptyText** if no contacts are available.

Adapter Class Implementation

Adapter Class (ContactAdapter)

- Binds a list of contacts to the RecyclerView.

ViewHolder Class (ContactViewHolder)

- Holds references to UI elements (name, phone, image, call button).

onCreateViewHolder

- Inflates the layout for each contact item.

onBindViewHolder

- Populates UI elements with contact data and sets click listeners for actions (call, edit).

Call Functionality

- Opens the phone dialer with the contact's number when the call button is clicked.

Edit Functionality

- Navigates to an EditContactActivity when a contact item is clicked.

```
class ContactAdapter(private val contacts: List<Contact>) :
    RecyclerView.Adapter<ContactAdapter.ContactViewHolder>() {
    Edit | Explain | Test | Document | Fix
    class ContactViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
        val name: TextView = itemView.findViewById(R.id.textViewName)
        val phone: TextView = itemView.findViewById(R.id.textViewPhone)
        val image: ImageView = itemView.findViewById(R.id.imageViewContact)
        private val call:ImageButton = itemView.findViewById(R.id.callButton)

        init {
            call.setOnClickListener {
                val intent = Intent(Intent.ACTION_DIAL, Uri.parse( uriString: "tel:${phone.text}"))
                it.context.startActivity(intent)
            }
        }
    }

    Edit | Explain | Test | Document | Fix
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ContactViewHolder {
        val view = LayoutInflater.from(parent.context)
            .inflate(R.layout.item_contact, parent, attachToRoot: false)
        return ContactViewHolder(view)
    }

    Edit | Explain | Test | Document | Fix
    override fun onBindViewHolder(holder: ContactViewHolder, position: Int) {
        val contact = contacts[position]
        holder.name.text = contact.name
        holder.phone.text = contact.phone

        holder.itemView.setOnClickListener {
            val intent = Intent(it.context,EditContactActivity::class.java)
            intent.putExtra( name: "id",contact.id)
            it.context.startActivity(intent)
        }
```

EditContactActivity

Binding (ActivityEditContactBinding)

- Simplifies access to the UI elements defined in the layout.

Contact ID (id)

- Retrieves the contact ID from the **Intent** to load specific contact details.

Selected Image Path (selectedImagePath)

- Stores the file path of the selected contact image.

onCreate Method

- Initializes the UI and retrieves contact details from the database using the provided ID.
- Populates the input fields (**editNameInput**, **editPhoneInput**) and loads the contact image:
  - Displays the contact's image if available.
  - Defaults to a placeholder image (**R.drawable.user**) if no image is set.

```
class EditContactActivity : AppCompatActivity() {

    private lateinit var binding: ActivityEditContactBinding
    private var id: Int = 0
    private var selectedImagePath: String? = ""
    Edit | Explain | Test | Document | Fix
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityEditContactBinding.inflate(layoutInflater)
        setContentView(binding.root)

        id = intent.getIntExtra( name: "id",  defaultValue: 0)
        val dbHelper = DatabaseHelper( context: this)
        val contact = dbHelper.getContactById(id)

        if (contact?.imagePath!!.isNotEmpty()) {
            loadImageFromFilePath(contact.imagePath, binding.editImageView)
            selectedImagePath = contact.imagePath
        } else {
            binding.editImageView.setImageResource(R.drawable.user)
        }

        binding.editNameInput.setText(contact.name)
        binding.editPhoneInput.setText(contact.phone)
```

EditContactActivity

```kotlin
// Edit Contact
binding.editButton.setOnClickListener {
    val name = binding.editNameInput.text.toString()
    val phone = binding.editPhoneInput.text.toString()
    if (name.isNotEmpty() && phone.isNotEmpty()) {
        val contact =
            Contact(id = id, name = name, phone = phone, imagePath = selectedImagePath!!)
        val result = dbHelper.updateContact(contact)
        if (result > 0) {
            Toast.makeText( context: this, text: "Contact edited successfully!", Toast.LENGTH_SHORT).show()
            finish()
            val intent = Intent( packageContext: this, MainActivity::class.java)
            startActivity(intent)
        } else {
            Toast.makeText( context: this, text: "Failed to edit contact!", Toast.LENGTH_SHORT).show()
        }
    } else {
        Toast.makeText( context: this, text: "All fields are required!", Toast.LENGTH_SHORT).show()
    }
}

binding.deleteButton.setOnClickListener {
    val delete = dbHelper.deleteContact(id)

    if (delete.toString().isNotEmpty()) {
        Toast.makeText( context: this, text: "Contact deleted successfully!", Toast.LENGTH_SHORT).show()
        finish()
        val intent = Intent( packageContext: this, MainActivity::class.java)
        startActivity(intent)
    } else {
        Toast.makeText( context: this, text: "Delete failed!", Toast.LENGTH_SHORT).show()

    }
```

Editing and Deleting Contact Details

Edit Button Functionality (editButton)

- Validates that both the name and phone fields are filled.
- Updates the contact in the database using **updateContact**.
- Displays a success or failure message using **Toast**.
- Navigates back to the **MainActivity** upon successful editing.

Delete Button Functionality (deleteButton)

- Deletes the contact from the database using **deleteContact**.
- Shows a success or failure message using **Toast**.
- Returns to the **MainActivity** after a successful deletion.

Thank You