Module 6: Portfolio Milestone

Mohammad Sargazi

Colorado State University Global

CSC475: Platform-Based Development

Dr. Bari

Oct 30, 2024

1- Challenges

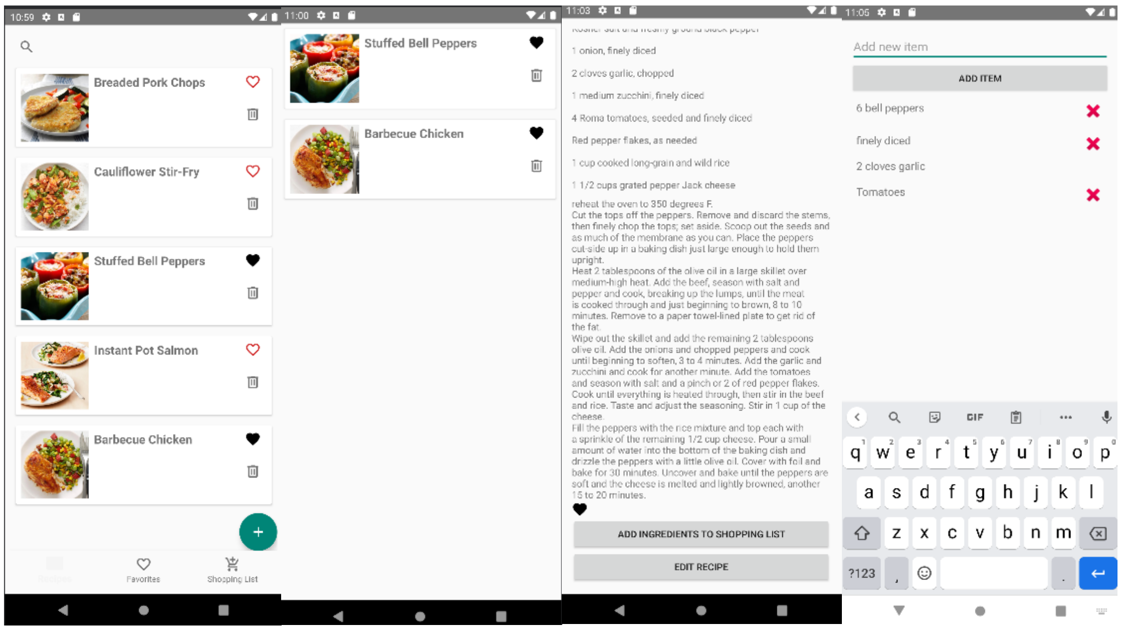
Working on this project has been a true learning experience, pushing me to explore areas of Android development that I hadn’t encountered before. Right from the start, designing the app's structure required a lot of thought, particularly when it came to organizing the activities and fragments for each function. Integrating the RecyclerView with a custom adapter was another step that took time, as getting it to display all the recipe information consistently wasn’t as straightforward as I initially thought. Each component, from the adapter to the database helper, had to work together seamlessly, which challenged me to understand how each part fits into the bigger picture of app functionality.

Database management was one of the most challenging parts, as it involved both retrieving and manipulating data in real-time. Establishing the SQLite database and handling user inputs presented various unexpected issues, especially when it came to updating and deleting items without impacting the overall app flow. Creating methods to update favorite recipes or remove items from the shopping list allowed me to understand database transactions better and make sure data handling was efficient and secure. It was rewarding to see the app properly display updated information after these functions were implemented.

The design of the user interface also offered valuable insights into making an app intuitive and user-friendly. Adding search functionality to the app's main screen required me to apply filtering techniques that would allow users to find specific recipes quickly. The search functionality was one of the pieces that brought all parts of the project together, combining UI elements with database queries. Every layout file had to be carefully structured to accommodate different types of user interaction, from viewing and editing recipes to navigating between various screens. This required attention to detail to ensure the app looked good and functioned well.

Exploring UI icons for edit, delete, and favorite features brought an unexpected element of creativity to the project. Finding icons that fit the app's theme added a layer of personalization, making the app feel more polished. This process also familiarized me with the Android resources system and how to manage drawable assets effectively. Balancing functionality with aesthetics was definitely a rewarding part of the project, and it taught me how even small design choices can impact user experience significantly. This experience has definitely strengthened my skills in Android development and prepared me for tackling even more complex projects in the future.

2- Screenshots



Kosher salt and freshly ground black pepper

1 onion, finely diced

2 cloves garlic, chopped

1 medium zucchini, finely diced

4 Roma tomatoes, seeded and finely diced

Red pepper flakes, as needed

1 cup cooked long-grain and wild rice

1 1/2 cups grated pepper Jack cheese

reheat the oven to 350 degrees F.
Cut the tops off the peppers. Remove and discard the stems, then finely chop the tops; set aside. Scoop out the seeds and as much of the membrane as you can. Place the peppers cut-side up in a baking dish just large enough to hold them upright.
Heat 2 tablespoons of the olive oil in a large skillet over medium-high heat. Add the beef, season with salt and pepper and cook, breaking up the lumps, until the meat is cooked through and just beginning to brown, 8 to 10 minutes. Remove to a paper towel-lined plate to get rid of the fat.
Wipe out the skillet and add the remaining 2 tablespoons olive oil. Add the onions and chopped peppers and cook until beginning to soften, 3 to 4 minutes. Add the garlic and zucchini and cook for another minute. Add the tomatoes and season with salt and a pinch or 2 of red pepper flakes. Cook until everything is heated through, then stir in the beef and rice. Taste and adjust the seasoning. Stir in 1 cup of the cheese.
Fill the peppers with the rice mixture and top each with a sprinkle of the remaining 1/2 cup cheese. Pour a small amount of water into the bottom of the baking dish and drizzle the peppers with a little olive oil. Cover with foil and bake for 30 minutes. Uncover and bake until the peppers are soft and the cheese is melted and lightly browned, another 15 to 20 minutes.

3- Source codes:

3.1- AddRecipeActivity.kt

```
package com.example.recipe

import android.Manifest
import android.app.Activity
import android.content.Intent
import android.content.pm.PackageManager
import android.net.Uri
import android.os.Build
import android.os.Bundle
import android.provider.OpenableColumns
import android.widget.Toast
import androidx.activity.result.ActivityResultLauncher
import androidx.activity.result.contract.ActivityResultContracts
import androidx.appcompat.app.AppCompatActivity
import androidx.core.content.ContextCompat
import com.example.recipe.databinding.ActivityAddRecipeBinding

/**
 * Activity to add a new recipe. Allows the user to input recipe details,
 * select an image, and save the recipe to the database.
 */
class AddRecipeActivity : AppCompatActivity() {

    private lateinit var binding: ActivityAddRecipeBinding // View binding to
access UI elements
    private var selectedImageUri: Uri? = null // Variable to store selected
image URI
    private lateinit var imagePickerLauncher: ActivityResultLauncher<Intent>
// Launcher for picking an image
    private lateinit var databaseHelper: DatabaseHelper // Helper to interact
with the database
    private lateinit var requestPermissionLauncher:
ActivityResultLauncher<String> // Launcher for requesting permissions

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityAddRecipeBinding.inflate(layoutInflater)
        setContentView(binding.root)

        // Initialize the database helper
        databaseHelper = DatabaseHelper(this)

        // Initialize the image picker launcher to handle image selection
result
        imagePickerLauncher =
registerForActivityResult(ActivityResultContracts.StartActivityForResult()) {
result ->
            if (result.resultCode == Activity.RESULT_OK && result.data !=
null) {
                // Successfully selected an image
                selectedImageUri = result.data?.data
                selectedImageUri?.let {
                    // Display the selected image in the ImageView
                    binding.ivRecipeImage.setImageURI(it)
                }
            } else {
                // Handle case when image selection was cancelled
                Toast.makeText(this, "Image selection cancelled",
Toast.LENGTH_SHORT).show()
            }
        }

        // Initialize the permission request launcher for storage permissions
        requestPermissionLauncher =
registerForActivityResult(ActivityResultContracts.RequestPermission()) {
isGranted ->
```

```
            if (isGranted) {
                // Permission granted, proceed to pick an image
                pickImageFromGallery()
            } else {
                // Permission denied, show a message
                Toast.makeText(this, "Permission denied",
Toast.LENGTH_SHORT).show()
            }
        }

        // Set a click listener to trigger permission check and image
selection
        binding.btnUploadImage.setOnClickListener {
            checkAndRequestPermission()
        }

        // Set a click listener for saving the recipe to the database
        binding.btnSaveRecipe.setOnClickListener {
            saveRecipe()
        }
    }

    /**
     * Checks and requests necessary permissions based on Android version.
     */
    private fun checkAndRequestPermission() {
        // Determine the permission based on Android version
        val permission = if (Build.VERSION.SDK_INT >=
Build.VERSION_CODES.TIRAMISU) {
            Manifest.permission.READ_MEDIA_IMAGES // Newer permission for
media images
        } else {
            Manifest.permission.READ_EXTERNAL_STORAGE // Legacy permission
for external storage
        }

        when {
            // Check if permission is already granted
            ContextCompat.checkSelfPermission(this, permission) ==
PackageManager.PERMISSION_GRANTED -> {
                // Permission already granted, proceed to pick an image
                pickImageFromGallery()
            }
            // Show rationale if permission was denied previously
            shouldShowRequestPermissionRationale(permission) -> {
                Toast.makeText(this, "Permission required to access images",
Toast.LENGTH_LONG).show()
                requestPermissionLauncher.launch(permission) // Request
permission again
            }
            else -> {
                // Directly request permission if it's the first time
                requestPermissionLauncher.launch(permission)
            }
        }
    }

    /**
```

```
     * Launches the system's image picker for the user to select an image.
     */
    private fun pickImageFromGallery() {
        val intent = Intent(Intent.ACTION_OPEN_DOCUMENT).apply {
            addCategory(Intent.CATEGORY_OPENABLE) // Allows the user to
select a file that is openable
            type = "image/*" // Limits selection to images
        }
        imagePickerLauncher.launch(intent) // Start the image picker activity
    }

    /**
     * Saves the recipe details entered by the user to the database.
     */
    private fun saveRecipe() {
        // Get user input for recipe name, ingredients, and notes
        val recipeName = binding.etRecipeName.text.toString().trim()
        val ingredients = binding.etIngredients.text.toString().trim()
        val notes = binding.etNotes.text.toString().trim()

        // Validate that all fields are filled
        if (recipeName.isEmpty() || ingredients.isEmpty() || notes.isEmpty())
{
            Toast.makeText(this, "Please fill all fields",
Toast.LENGTH_SHORT).show()
            return // Exit the function if any field is empty
        }

        // Ensure that an image has been selected
        if (selectedImageUri == null) {
            Toast.makeText(this, "Please select an image",
Toast.LENGTH_SHORT).show()
            return // Exit the function if no image is selected
        }

        // Create a new Recipe object with the entered details
        val newRecipe = Recipe(
            id = 0, // ID will be auto-generated by the database
            name = recipeName,
            ingredients = ingredients,
            notes = notes,
            imageUri = selectedImageUri.toString(), // Save the URI of the
selected image
            isFavorite = false // Default the recipe as non-favorite
        )

        // Insert the new recipe into the database
        val success = databaseHelper.insertRecipe(newRecipe)

        if (success) {
            // Display success message and close the activity
            Toast.makeText(this, "Recipe added successfully",
Toast.LENGTH_SHORT).show()
            finish()
        } else {
            // Display failure message if insertion fails
            Toast.makeText(this, "Failed to add recipe",
Toast.LENGTH_SHORT).show()
```

```
        }
    }
}
```

3.2- DatabaseHelper.kt

```
package com.example.recipe

import android.content.ContentValues
import android.content.Context
import android.database.sqlite.SQLiteDatabase
import android.database.sqlite.SQLiteOpenHelper

/**
 * DatabaseHelper is responsible for creating, managing, and interacting with
the SQLite database.
 * It includes methods for adding, retrieving, updating, and deleting recipes
and shopping list items.
 */
class DatabaseHelper(context: Context) : SQLiteOpenHelper(context,
"recipes.db", null, 2) {

    // This method is called the first time the database is created
    override fun onCreate(db: SQLiteDatabase) {
        // Create the "recipes" table to store all recipe data
        db.execSQL(
            """
            CREATE TABLE recipes (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                name TEXT,
                ingredients TEXT,
                notes TEXT,
                imageUri TEXT,
                isFavorite INTEGER DEFAULT 0
            )
            """
        )

        // Create the "shopping_list" table to store ingredients the user
adds to the shopping list
        db.execSQL(
            """
            CREATE TABLE shopping_list (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                item TEXT
            )
            """
        )

        // Insert a sample recipe to demonstrate how recipes are stored
        insertSampleRecipes(db)
    }

    /**
     * Inserts a sample recipe into the database. This is useful for testing
```

```
and for demonstrating the app to users.
     */
    private fun insertSampleRecipes(db: SQLiteDatabase) {
        val sampleRecipe = Recipe(
            id = 0, // ID will be set automatically by the database
            name = "Breaded Pork Chops",
            ingredients = "Pork chops, bread crumbs, eggs, seasoning",
            notes = "Delicious crispy pork chops.",
            imageUri =
"android.resource://com.example.recipe/drawable/breaded_pork_chops", //
Placeholder image URI
            isFavorite = false // Default to non-favorite
        )
        insertRecipe(db, sampleRecipe)
    }

    /**
     * Inserts a recipe into the "recipes" table.
     * @param db SQLiteDatabase instance to interact with the database.
     * @param recipe Recipe object containing details to insert into the
table.
     */
    private fun insertRecipe(db: SQLiteDatabase, recipe: Recipe) {
        val values = ContentValues().apply {
            put("name", recipe.name)
            put("ingredients", recipe.ingredients)
            put("notes", recipe.notes)
            put("imageUri", recipe.imageUri)
            put("isFavorite", if (recipe.isFavorite) 1 else 0)
        }
        db.insert("recipes", null, values) // Insert the recipe data into the
"recipes" table
    }

    // Called when the database version changes, to upgrade the database
structure
    override fun onUpgrade(db: SQLiteDatabase, oldVersion: Int, newVersion:
Int) {
        db.execSQL("DROP TABLE IF EXISTS recipes") // Drop old "recipes"
table
        db.execSQL("DROP TABLE IF EXISTS shopping_list") // Drop old
"shopping_list" table
        onCreate(db) // Recreate the tables by calling onCreate
    }

    /**
     * Inserts a new recipe into the database.
     * @param recipe Recipe object to insert.
     * @return Boolean indicating success or failure of the insert operation.
     */
    fun insertRecipe(recipe: Recipe): Boolean {
        val db = writableDatabase
        val values = ContentValues().apply {
            put("name", recipe.name)
            put("ingredients", recipe.ingredients)
            put("notes", recipe.notes)
            put("imageUri", recipe.imageUri)
            put("isFavorite", if (recipe.isFavorite) 1 else 0)
        }
        return db.insert("recipes", null, values) != -1L
    }

    /**
     * Retrieves a recipe by its ID from the database.
     * @param id ID of the recipe to retrieve.
     * @return Recipe object if found, or null if not found.
     */
    fun getRecipeById(id: Int): Recipe? {
        val db = readableDatabase
        val cursor = db.query("recipes", null, "id = ?",
arrayOf(id.toString()), null, null, null)

        return if (cursor.moveToFirst()) {
            // Build a Recipe object from the data in the database
            val recipe = Recipe(
                id = cursor.getInt(cursor.getColumnIndexOrThrow("id")),
                name =
cursor.getString(cursor.getColumnIndexOrThrow("name")),
                ingredients =
cursor.getString(cursor.getColumnIndexOrThrow("ingredients")),
                notes =
cursor.getString(cursor.getColumnIndexOrThrow("notes")),
                imageUri =
cursor.getString(cursor.getColumnIndexOrThrow("imageUri")),
                isFavorite =
cursor.getInt(cursor.getColumnIndexOrThrow("isFavorite")) == 1
            )
            cursor.close()
            recipe
        } else {
            cursor.close()
            null // Return null if no recipe was found
        }
    }

    /**
     * Retrieves all recipes from the database.
     * @return List of all Recipe objects in the database.
     */
    fun getAllRecipes(): List<Recipe> {
        val db = readableDatabase
        val cursor = db.query("recipes", null, null, null, null, null, null)

        val recipes = mutableListOf<Recipe>()
        if (cursor.moveToFirst()) {
            do {
                recipes.add(
                    Recipe(
                        id =
cursor.getInt(cursor.getColumnIndexOrThrow("id")),
                        name =
cursor.getString(cursor.getColumnIndexOrThrow("name")),
                        ingredients =
cursor.getString(cursor.getColumnIndexOrThrow("ingredients")),
                        notes =
cursor.getString(cursor.getColumnIndexOrThrow("notes")),
                        imageUri =
cursor.getString(cursor.getColumnIndexOrThrow("imageUri")),
                        isFavorite =
cursor.getInt(cursor.getColumnIndexOrThrow("isFavorite")) == 1
                    )
                )
            } while (cursor.moveToNext())
        }
        cursor.close()
        return recipes // Return the list of recipes
    }

    /**
     * Retrieves only the recipes marked as favorites.
     * @return List of favorite recipes.
     */
    fun getFavoriteRecipes(): List<Recipe> {
        val db = readableDatabase
        val cursor = db.query(
            "recipes",
            null,
            "isFavorite = ?",
            arrayOf("1"),
            null,
            null,
            null
        )

        val favoriteRecipes = mutableListOf<Recipe>()
        if (cursor.moveToFirst()) {
            do {
                favoriteRecipes.add(
                    Recipe(
                        id =
cursor.getInt(cursor.getColumnIndexOrThrow("id")),
                        name =
cursor.getString(cursor.getColumnIndexOrThrow("name")),
                        ingredients =
cursor.getString(cursor.getColumnIndexOrThrow("ingredients")),
                        notes =
cursor.getString(cursor.getColumnIndexOrThrow("notes")),
                        imageUri =
cursor.getString(cursor.getColumnIndexOrThrow("imageUri")),
                        isFavorite =
cursor.getInt(cursor.getColumnIndexOrThrow("isFavorite")) == 1
                    )
                )
            } while (cursor.moveToNext())
        }
        cursor.close()
        return favoriteRecipes
    }

    /**
     * Updates an existing recipe in the database.
     * @param recipe Recipe object with updated data.
     * @return Boolean indicating success of the update.
     */
    fun updateRecipe(recipe: Recipe): Boolean {
        val db = writableDatabase
        val values = ContentValues().apply {
            put("name", recipe.name)
            put("ingredients", recipe.ingredients)
            put("notes", recipe.notes)
            put("imageUri", recipe.imageUri)
            put("isFavorite", if (recipe.isFavorite) 1 else 0)
        }
        return db.update("recipes", values, "id = ?",
arrayOf(recipe.id.toString())) > 0
    }

    /**
     * Toggles the favorite status of a recipe.
     * @param recipeId ID of the recipe to update.
     * @param isFavorite New favorite status.
     * @return Boolean indicating success of the update.
     */
    fun updateRecipeFavoriteStatus(recipeId: Int, isFavorite: Boolean):
Boolean {
        val db = writableDatabase
        val values = ContentValues().apply {
            put("isFavorite", if (isFavorite) 1 else 0)
        }
        return db.update("recipes", values, "id = ?",
arrayOf(recipeId.toString())) > 0
    }

    /**
     * Retrieves all items in the shopping list.
     * @return List of shopping list items as strings.
     */
    fun getShoppingListItems(): List<String> {
        val db = readableDatabase
        val cursor = db.query("shopping_list", null, null, null, null, null,
null)

        val items = mutableListOf<String>()
        if (cursor.moveToFirst()) {
            do {

items.add(cursor.getString(cursor.getColumnIndexOrThrow("item")))
            } while (cursor.moveToNext())
        }
        cursor.close()
        return items
    }

    /**
     * Inserts an item into the shopping list.
     * @param item String representing the item to add.
     * @return Boolean indicating success of the insertion.
     */
    fun insertShoppingListItem(item: String): Boolean {
        val db = writableDatabase
        val values = ContentValues().apply {
            put("item", item)
        }
        return db.insert("shopping_list", null, values) != -1L
    }

    /**
     * Deletes an item from the shopping list.
     * @param item String representing the item to delete.
     * @return Boolean indicating success of the deletion.
     */
    fun deleteShoppingListItem(item: String): Boolean {
        val db = writableDatabase
        return db.delete("shopping_list", "item = ?", arrayOf(item)) > 0
    }

    /**
     * Deletes a recipe from the database by ID.
     * @param id ID of the recipe to delete.
     * @return Boolean indicating success of the deletion.
     */
    fun deleteRecipe(id: Int): Boolean {
        val db = writableDatabase
        val result = db.delete("recipes", "id = ?", arrayOf(id.toString()))
        return result > 0
    }
}
```

3.3- EditRecipeActivity.kt

```
package com.example.recipe

import android.app.Activity
import android.content.Intent
import android.net.Uri
import android.os.Bundle
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import com.example.recipe.databinding.ActivityEditRecipeBinding

class EditRecipeActivity : AppCompatActivity() {

    private lateinit var binding: ActivityEditRecipeBinding  // View binding
for layout access
    private lateinit var databaseHelper: DatabaseHelper  // Helper for
interacting with the database
    private var recipeId: Int = -1  // ID of the recipe being edited
    private var imageUri: Uri? = null  // URI for the selected recipe image

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityEditRecipeBinding.inflate(layoutInflater)
        setContentView(binding.root)

        databaseHelper = DatabaseHelper(this)
        recipeId = intent.getIntExtra("RECIPE_ID", -1)
        loadRecipeDetails()
        binding.ivRecipeImage.setOnClickListener { openImagePicker() }
        binding.btnSave.setOnClickListener { saveRecipeChanges() }
    }

    private fun openImagePicker() {
        val intent = Intent(Intent.ACTION_OPEN_DOCUMENT).apply {
            addCategory(Intent.CATEGORY_OPENABLE)
            type = "image/*"  // Only allow images to be selected
        }
        startActivityForResult(intent, REQUEST_CODE_PICK_IMAGE)
    }

    override fun onActivityResult(requestCode: Int, resultCode: Int, data:
Intent?) {
        super.onActivityResult(requestCode, resultCode, data)
        if (requestCode == REQUEST_CODE_PICK_IMAGE && resultCode ==
Activity.RESULT_OK) {
            imageUri = data?.data  // Store the selected image URI
            imageUri?.let { uri ->
                try {
                    // Request persistable permission to use the URI across
sessions
                    contentResolver.takePersistableUriPermission(uri,
Intent.FLAG_GRANT_READ_URI_PERMISSION)
                    binding.ivRecipeImage.setImageURI(uri)  // Display the
selected image
                } catch (e: SecurityException) {
                    Toast.makeText(this, "Permission denied for this image",
Toast.LENGTH_SHORT).show()
                }
            } ?: run {
                Toast.makeText(this, "Failed to load image",
Toast.LENGTH_SHORT).show()
            }
        }
    }

    private fun loadRecipeDetails() {
        val recipe = databaseHelper.getRecipeById(recipeId)
        recipe?.let {
            binding.etRecipeName.setText(it.name)
            binding.etIngredients.setText(it.ingredients)
            binding.etNotes.setText(it.notes)

            if (!it.imageUri.isNullOrEmpty()) {
                val uri = Uri.parse(it.imageUri)
                binding.ivRecipeImage.setImageURI(uri)
            } else {

binding.ivRecipeImage.setImageResource(R.drawable.ic_no_image)
            }
        } ?: run {
            Toast.makeText(this, "Recipe not found",
Toast.LENGTH_SHORT).show()
            finish()
        }
    }
    private fun saveRecipeChanges() {
        val name = binding.etRecipeName.text.toString().trim()
        val ingredients = binding.etIngredients.text.toString().trim()
        val notes = binding.etNotes.text.toString().trim()

        if (name.isEmpty() || ingredients.isEmpty() || notes.isEmpty()) {
            Toast.makeText(this, "Please fill all fields",
Toast.LENGTH_SHORT).show()
            return
        }

        val updatedRecipe = Recipe(
            id = recipeId,
            name = name,
            ingredients = ingredients,
            notes = notes,
            imageUri = imageUri?.toString() ?: "",
            isFavorite = false
        )

        if (databaseHelper.updateRecipe(updatedRecipe)) {
            Toast.makeText(this, "Recipe updated", Toast.LENGTH_SHORT).show()
            finish()
        } else {
            Toast.makeText(this, "Failed to update recipe",
Toast.LENGTH_SHORT).show()
        }
    }

    companion object {
        const val REQUEST_CODE_PICK_IMAGE = 100  // Request code for image
picker
    }
}
```

3.4- FavoritesActivity.kt

```
// FavoritesActivity.kt
package com.example.recipe

import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity
import androidx.recyclerview.widget.LinearLayoutManager
import com.example.recipe.databinding.ActivityFavoritesBinding

class FavoritesActivity : AppCompatActivity() {

    private lateinit var binding: ActivityFavoritesBinding
    private lateinit var databaseHelper: DatabaseHelper
    private lateinit var recipeAdapter: RecipeAdapter
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityFavoritesBinding.inflate(layoutInflater)
        setContentView(binding.root)

        databaseHelper = DatabaseHelper(this)

        recipeAdapter = RecipeAdapter(
            this,
            databaseHelper.getFavoriteRecipes(),
            onRecipeClick = { recipe -> /* Handle click */ },
            onEditClick = { recipe -> /* Handle edit */ },
            onDeleteClick = { recipe ->
                databaseHelper.deleteRecipe(recipe.id)
                loadFavoriteRecipes()
            },
            onFavoriteClick = { recipe ->
                val isFavorite = !recipe.isFavorite
                recipe.isFavorite = isFavorite
                databaseHelper.updateRecipeFavoriteStatus(recipe.id,
isFavorite)
                loadFavoriteRecipes()
            }
        )

        binding.favoritesRecyclerView.layoutManager =
LinearLayoutManager(this)
        binding.favoritesRecyclerView.adapter = recipeAdapter

        loadFavoriteRecipes()
    }

    private fun loadFavoriteRecipes() {
        val favorites = databaseHelper.getFavoriteRecipes()
        recipeAdapter.updateList(favorites)
    }
}
```

3.5- MainActivity.kt

```
package com.example.recipe

import android.content.Intent
import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity
import androidx.recyclerview.widget.LinearLayoutManager
import com.example.recipe.databinding.ActivityMainBinding
import androidx.appcompat.widget.SearchView

class MainActivity : AppCompatActivity() {

    private lateinit var binding: ActivityMainBinding  // Binding to access
layout views
    private lateinit var databaseHelper: DatabaseHelper  // Database helper
```

```
instance
    private lateinit var recipeAdapter: RecipeAdapter  // Adapter to handle
recipe list
    private var recipes: MutableList<Recipe> = mutableListOf()  // All
recipes from the database
    private var filteredRecipes: MutableList<Recipe> = mutableListOf()  //
Filtered recipe list

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        databaseHelper = DatabaseHelper(this)

        // Load all recipes from the database
        recipes = databaseHelper.getAllRecipes().toMutableList()
        filteredRecipes.addAll(recipes)

        // Setup RecyclerView with the adapter
        recipeAdapter = RecipeAdapter(
            context = this,
            recipes = filteredRecipes,
            onRecipeClick = { recipe ->  // Handle recipe item click
                val intent = Intent(this,
ViewRecipeActivity::class.java).apply {
                    putExtra("RECIPE_ID", recipe.id)  // Pass the recipe ID
to view activity
                }
                startActivity(intent)
            },
            onEditClick = { recipe ->  // Handle edit button click
                val intent = Intent(this,
EditRecipeActivity::class.java).apply {
                    putExtra("RECIPE_ID", recipe.id)  // Pass the recipe ID
to edit activity
                }
                startActivity(intent)
            },
            onDeleteClick = { recipe ->  // Handle delete button click
                databaseHelper.deleteRecipe(recipe.id)
                refreshRecipeList()  // Refresh the list after deletion
            },
            onFavoriteClick = { recipe ->  // Handle favorite toggle click
                recipe.isFavorite = !recipe.isFavorite
                databaseHelper.updateRecipeFavoriteStatus(recipe.id,
recipe.isFavorite)
                refreshRecipeList()
            }
        )

        binding.recipeRecyclerView.layoutManager = LinearLayoutManager(this)
        binding.recipeRecyclerView.adapter = recipeAdapter

        // Floating action button to add a new recipe
        binding.fabAddRecipe.setOnClickListener {
            val intent = Intent(this, AddRecipeActivity::class.java)
            startActivity(intent)
        }

        // Search view to filter recipes by name
        binding.searchView.setOnQueryTextListener(object :
SearchView.OnQueryTextListener {
            override fun onQueryTextSubmit(query: String?): Boolean {
                filterRecipes(query.orEmpty())
                return true
            }

            override fun onQueryTextChange(newText: String?): Boolean {
                filterRecipes(newText.orEmpty())
                return true
            }
        })

        // Handle bottom navigation clicks
        binding.bottomNavigationView.setOnItemSelectedListener { item ->
            when (item.itemId) {
                R.id.nav_recipes -> true
                R.id.nav_favorites -> {
                    startActivity(Intent(this,
FavoritesActivity::class.java))
                    true
                }
                R.id.nav_shopping_list -> {
                    startActivity(Intent(this,
ShoppingListActivity::class.java))
                    true
                }
                else -> false
            }
        }
    }

    // Filter recipes based on the search query
    private fun filterRecipes(query: String) {
        filteredRecipes.clear()
        if (query.isEmpty()) {
            filteredRecipes.addAll(recipes)
        } else {
            filteredRecipes.addAll(recipes.filter {
                it.name.contains(query, ignoreCase = true)
            })
        }
        recipeAdapter.notifyDataSetChanged()
    }

    // Refresh the recipe list from the database
    private fun refreshRecipeList() {
        recipes.clear()
        recipes.addAll(databaseHelper.getAllRecipes())
        filteredRecipes.clear()
        filteredRecipes.addAll(recipes)
        recipeAdapter.notifyDataSetChanged()
    }

    // Refresh the list when the activity resumes
    override fun onResume() {
        super.onResume()
        refreshRecipeList()
    }
}
```

3.6- Recipe.kt

```
package com.example.recipe

data class Recipe(
    val id: Int = 0,  // Auto-generated by the database
    val name: String,
    val ingredients: String,
    val notes: String,
    val imageUri: String,  // Use URI instead of blob
    var isFavorite: Boolean
)
```

3.7- RecipeAdapter.kt

```
package com.example.recipe

import android.content.Context
import android.net.Uri
import android.util.Log
import android.view.LayoutInflater
import android.view.ViewGroup
import androidx.recyclerview.widget.RecyclerView
import com.bumptech.glide.Glide
import com.example.recipe.databinding.ItemRecipeBinding

class RecipeAdapter(
    private val context: Context,
    private var recipes: List<Recipe>,
    private val onRecipeClick: (Recipe) -> Unit,
    private val onEditClick: (Recipe) -> Unit,
    private val onDeleteClick: (Recipe) -> Unit,
    private val onFavoriteClick: (Recipe) -> Unit
) : RecyclerView.Adapter<RecipeAdapter.RecipeViewHolder>() {

    inner class RecipeViewHolder(private val binding: ItemRecipeBinding) :
        RecyclerView.ViewHolder(binding.root) {

        fun bind(recipe: Recipe) {
            binding.tvRecipeName.text = recipe.name

            try {
                Glide.with(context)
                    .load(Uri.parse(recipe.imageUri))
                    .placeholder(R.drawable.ic_no_image)
                    .error(R.drawable.ic_no_image)
                    .into(binding.ivRecipeImage)
            } catch (e: Exception) {
                Log.e("RecipeAdapter", "Error loading image URI:
${recipe.imageUri}", e)

binding.ivRecipeImage.setImageResource(R.drawable.ic_no_image)
            }
            // Update heart icon based on favorite status
            val favoriteIcon = if (recipe.isFavorite) R.drawable.ic_favorite
else R.drawable.ic_favorite_border
            binding.btnFavorite.setImageResource(favoriteIcon)

            // Set click listeners
            binding.root.setOnClickListener { onRecipeClick(recipe) }
            binding.btnEdit.setOnClickListener { onEditClick(recipe) }
            binding.btnDelete.setOnClickListener { onDeleteClick(recipe) }

            binding.btnFavorite.setOnClickListener {
                onFavoriteClick(recipe)
            }
        }
    }
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
RecipeViewHolder {
        val binding =
ItemRecipeBinding.inflate(LayoutInflater.from(parent.context), parent, false)
        return RecipeViewHolder(binding)
    }

    override fun onBindViewHolder(holder: RecipeViewHolder, position: Int) {
        holder.bind(recipes[position])
    }

    override fun getItemCount(): Int = recipes.size

    fun updateList(newRecipes: List<Recipe>) {
        recipes = newRecipes
        notifyDataSetChanged()
    }
}
```

3.8- ShoppingListActivity.kt

```
// ShoppingListActivity.kt
package com.example.recipe

import android.os.Bundle
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import androidx.recyclerview.widget.LinearLayoutManager
import com.example.recipe.databinding.ActivityShoppingListBinding

class ShoppingListActivity : AppCompatActivity() {

    private lateinit var binding: ActivityShoppingListBinding
    private lateinit var dbHelper: DatabaseHelper
    private lateinit var shoppingListAdapter: ShoppingListAdapter
```

```kotlin
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityShoppingListBinding.inflate(layoutInflater)
        setContentView(binding.root)

        dbHelper = DatabaseHelper(this)

        setupRecyclerView()

        binding.btnAddItem.setOnClickListener {
            val newItem = binding.etNewItem.text.toString().trim()
            if (newItem.isNotEmpty()) {
                dbHelper.insertShoppingListItem(newItem)
                binding.etNewItem.text.clear()
                loadShoppingListItems()
            } else {
                Toast.makeText(this, "Please enter an item",
Toast.LENGTH_SHORT).show()
            }
        }

        loadShoppingListItems()
    }

    private fun setupRecyclerView() {
        shoppingListAdapter = ShoppingListAdapter(mutableListOf()) { item ->
            dbHelper.deleteShoppingListItem(item)
            loadShoppingListItems()
        }
        binding.shoppingListRecyclerView.layoutManager =
LinearLayoutManager(this)
        binding.shoppingListRecyclerView.adapter = shoppingListAdapter
    }

    private fun loadShoppingListItems() {
        val items = dbHelper.getShoppingListItems()
        shoppingListAdapter.updateList(items)
    }
}
```

3.9 – ShoppingListAdapter

```kotlin
package com.example.recipe

import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.TextView
import androidx.appcompat.widget.AppCompatImageButton
import androidx.recyclerview.widget.RecyclerView
```

```
class ShoppingListAdapter(
    private var items: MutableList<String>,
    private val onDeleteClick: (String) -> Unit
) : RecyclerView.Adapter<ShoppingListAdapter.ShoppingListViewHolder>() {

    inner class ShoppingListViewHolder(view: View) :
RecyclerView.ViewHolder(view) {
        val tvItem: TextView = view.findViewById(R.id.tvItem)
        val btnDelete: AppCompatImageButton =
view.findViewById(R.id.btnDelete)
    }
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
ShoppingListViewHolder {
        val view = LayoutInflater.from(parent.context)
            .inflate(R.layout.item_shopping_list, parent, false)
        return ShoppingListViewHolder(view)
    }

    override fun getItemCount(): Int = items.size

    override fun onBindViewHolder(holder: ShoppingListViewHolder, position:
Int) {
        val item = items[position]
        holder.tvItem.text = item

        holder.btnDelete.setOnClickListener {
            onDeleteClick(item)
            removeItem(position)
        }
    }
    // Method to update the list of items
    fun updateList(newItems: List<String>) {
        items.clear()
        items.addAll(newItems)
        notifyDataSetChanged()
    }
    // Helper method to remove an item from the list
    private fun removeItem(position: Int) {
        if (position >= 0 && position < items.size) {
            items.removeAt(position)
            notifyItemRemoved(position)
            notifyItemRangeChanged(position, items.size)
        }
    }
}
```

3.10- ViewRecipeActivity.kt

```
// ViewRecipeActivity.kt
package com.example.recipe

import android.content.Intent
import android.net.Uri
import android.os.Bundle
```

```kotlin
import android.util.Log
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import com.example.recipe.databinding.ActivityViewRecipeBinding

class ViewRecipeActivity : AppCompatActivity() {

    private lateinit var binding: ActivityViewRecipeBinding  // View binding
for layout access
    private lateinit var dbHelper: DatabaseHelper  // Database helper
instance
    private var recipeId: Int = 0  // ID of the recipe to view
    private var recipe: Recipe? = null  // Recipe object to display
    private var isFavorite: Boolean = false  // Track favorite status

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityViewRecipeBinding.inflate(layoutInflater)
        setContentView(binding.root)

        dbHelper = DatabaseHelper(this)
        recipeId = intent.getIntExtra("RECIPE_ID", 0)

        loadRecipeData()

        binding.btnAddToFavorites.setOnClickListener { toggleFavorite() }
        binding.btnAddToShoppingList.setOnClickListener {
addIngredientsToShoppingList() }
        binding.btnEditRecipe.setOnClickListener {
            val intent = Intent(this, EditRecipeActivity::class.java)
            intent.putExtra("RECIPE_ID", recipeId)
            startActivity(intent)
        }
    }

    private fun loadRecipeData() {
        recipe = dbHelper.getRecipeById(recipeId)
        recipe?.let {
            binding.tvRecipeName.text = it.name
            binding.tvIngredients.text = it.ingredients
            binding.tvNotes.text = it.notes
            isFavorite = it.isFavorite

            if (!it.imageUri.isNullOrEmpty()) {
                val uri = Uri.parse(it.imageUri)
                try {
                    binding.ivRecipeImage.setImageURI(uri)  // Display the
image directly
                } catch (e: SecurityException) {
                    Log.e("ViewRecipeActivity", "Permission denied for URI:
${e.message}")

binding.ivRecipeImage.setImageResource(R.drawable.ic_no_image)
                    Toast.makeText(this, "Permission denied to load image",
Toast.LENGTH_SHORT).show()
                }
            } else {
```

```
binding.ivRecipeImage.setImageResource(R.drawable.ic_no_image)
            }

            updateFavoriteIcon()
        } ?: run {
            Toast.makeText(this, "Recipe not found",
Toast.LENGTH_SHORT).show()
            finish()
        }
    }

    private fun toggleFavorite() {
        isFavorite = !isFavorite
        dbHelper.updateRecipeFavoriteStatus(recipeId, isFavorite)
        updateFavoriteIcon()
    }

    private fun updateFavoriteIcon() {
        binding.btnAddToFavorites.setImageResource(
            if (isFavorite) R.drawable.ic_favorite else
R.drawable.ic_favorite_border
        )
    }

    private fun addIngredientsToShoppingList() {
        recipe?.ingredients?.split(",")?.map { it.trim() }?.forEach {
            dbHelper.insertShoppingListItem(it)
        }
        Toast.makeText(this, "Ingredients added to shopping list",
Toast.LENGTH_SHORT).show()
    }

    override fun onResume() {
        super.onResume()
        loadRecipeData()  // Refresh the data when returning to this activity
    }
}
```

References