DLBCSEMSE02 – Mobile Software Engineering II

Project Report

Task 3: Collecting memories – Develop a journal app

Wanderlog

GitHub Repository Link:
https://github.com/zdemirgm/20240517_Zehra_Demir_92130119_DLBCSEMSE02.git

Zehra Demir

Matriculation Number: 92130119

INTERNATIONAL
UNIVERSITY OF
APPLIED SCIENCES

Table of Contents

Introduction

In today's fast paced world where moments come and disappear like a flash, the necessity of retaining memories has never been more accentuated than in this era. No matter the reason, be it keeping track of a wonderful trip, recollecting personal accomplishments, or simply listing down everyday thoughts, journaling is a long standing tool for preserving the essence of life’s momentous occasions.

We realize that the value of this practice is paramount, upon which we start this journey of a journaling app (Wanderlog) development which is completely inclusive. Our ambition is to create a comprehensive tool for digital users to compile memories and enjoy their past experiences, merging the old style of diary keeping with the modern attribute of convenience.

Overview of Software Design:

The software design is a strategy used to manage complex operations, maintain performance, conserve energy, and protect the environment. The journal app (Wanderlog) focuses, among other things, on creating the basic functionality for writing down memories and thoughts, but what makes it stand out is its simplicity and user-friendliness. Attempts were done for the app's interface to be clear and simple, enhancing an enjoyable user experience. Furthermore, the application has the function to add, update, view and delete journal entries, with the ability to see details of each entry. Even though the app may not contain revolutionary innovations, the concentration on user-friendliness and efficiency makes the process of journaling fun and easy, and that is why it appeals to a diverse audience.

The journal app (Wanderlog) comes with a modular architecture where the business logic is separated from the presentation layer. This makes it more maintainable, highly scalable, and easy development. The primary components of the app and their interactions are as follows:

Components:

- Activities
- Data Models
- Database Helper
- Adapters
- Layouts

1. Activities

MainActivity: This is the point where the client will first launch the app. It displays the authentication of journal entries and provides buttons to add new entries and view pins.

EntryActivity: In this activity, users will be able to write a new piece or even edit an existing piece. Here, the model has fields for title, content, and date.

EntryDetailActivity: This activity demonstrates the details of an entry of a selected journal. The title, text and date are all shown on the activity. It is reminiscent of a feature that gives a user to either edit or delete the submissions.

PinsActivity: This activity also feature the list of pins that a user has achieved. It additionally displays a pin sharing button on many users' social media channels.

2. Data Models

JournalEntry: A java class that will act as a journal entry data. It has the following values: id, title, topic, and date.

3. DatabaseHelper

DatabaseHelper: The implementation of a singleton class that handles the database services. There are four operations which are available in this module: these are for adding, updating, retrieving, and deleting record.

4. Adapters

ArrayAdapter: The listview is bound to the list of journal entries using the journalListAdapter in MainActivity. It moves the information from the JournalEntry mode to the representable form that will be displayed.

5. Layouts

XML Layouts: Determine the UI for the every activity done. They explain how specific controls (buttons, text fields, lists) are placed on the screen.

Interactions Between Components:

MainActivity Interactions:

When MainActivity launches, the class obtains all journal entries from database by DatabaseHelper. getAllEntries() is called and the listview is loaded using an ArrayAdapter. By pressing the "Add Entry" button in MainActivity, we will see the EntryActivity screen for making a new entry.

Entering an entry in ListView moves to EntryDetailActivity via Intent with the entry id passed as parameter.

"Pins" button will lead to PinsActivity.

EntryActivity Interactions:

Enables users to type in the name, content, and date for their journal entry. Towards saving, it's insertion or update in the database. addEntry() or DatabaseHelper. updateEntry(). When the activity has been saved, it will return to MainActivity to show any new or updated entries made.

EntryDetailActivity Interactions:

Gets the journal entry details by means of the entry ID that were provided from MainActivity. Display the information and option for edit and delete the entry. Editing navigates to EntryActivity with the entry details given a head start. The Deleting method eliminates the entry from the database by DatabaseHelper. deleteEntry() and it goes back to MainActivity.

PinsActivity Interactions:

Displays what pins the user have achieved so far. The "Share Pins" button displays a list of social media platforms for sharing the pinned graphic.

Example Interaction Flow:

- Launching the App: MainActivity is launched. From it the journal entries are retrieved and displayed.
- Adding a New Entry: User taps "Add Entry" in MainActivity to create entry. At the start, Activity EntryActivity is used for making a new entry. User enters information and presses save button. Entry is inserted through database and EntryActivity now opens MainActivity again. The ListView is updated with new entry in MainActivity.
- Viewing an Entry: User taps on a specified item from the ListView that belongs to MainActivity. EntryDetailActivity is called with entries data. User can edit or delete the entry from here in EntryDetailActivity.
- Sharing Pins: User clicks on the "Pins" button of MainActivity. The PinsActivity class gets displayed with the list of pins being the main focus. A user clicks on share icons to initiate sharing options.

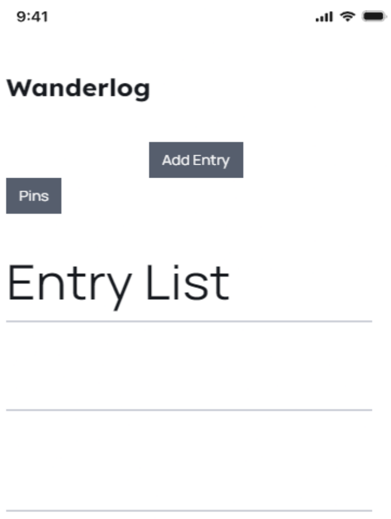
For my journal app, I used essential Android services and libraries in which the program was more functional and development was easy. SQLite acted as the main database management system enabling a full-proof and hassle-free solution for offline storage and local management of journal entries. Also, Android's in-built Intent system was employed for inter-component communication, which translated into smooth navigation across different screens and thus support users' interactions. To implement journal entries I used

ArrayAdapter to fill ListView with data from the SQLite database in a way that is seamless and with responsiveness in mind. The last but not the least Toast was incorporated to allow user notifications and feedback in the app thus the user experience was enhanced.

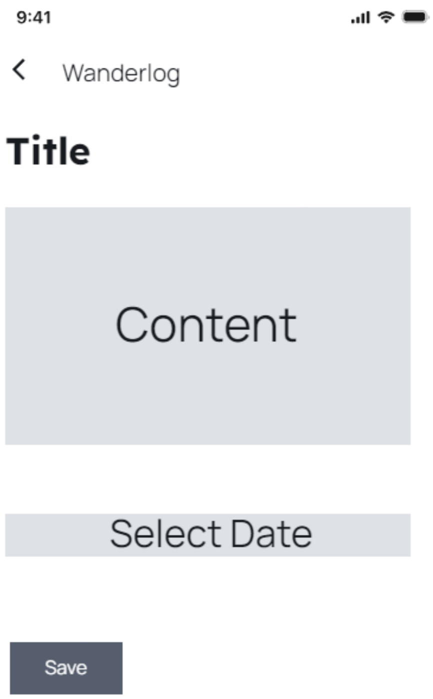
Wireframing Prensentation

Incorporating high fidelity wireframes into my project was pivotal for ensuring a visually appealing and functional user interface. These wireframes provided a detailed blueprint of the app's layout, navigation, and interactions, aligning with material design principles and Android app quality guidelines. Each screen, from entry creation to overview and detail views, was carefully designed to optimize usability and visual aesthetics, fostering an engaging user experience. These wireframes served as a visual guide throughout the development process, facilitating effective communication of design ideas and ensuring the final product meets user expectations.

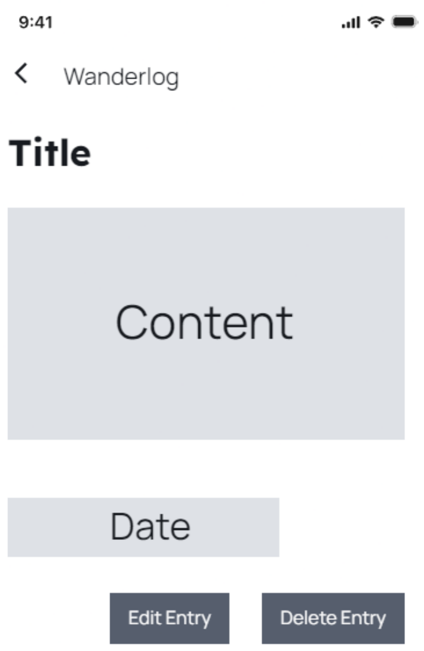
High-fidelity Wireframing Design for MainActivity



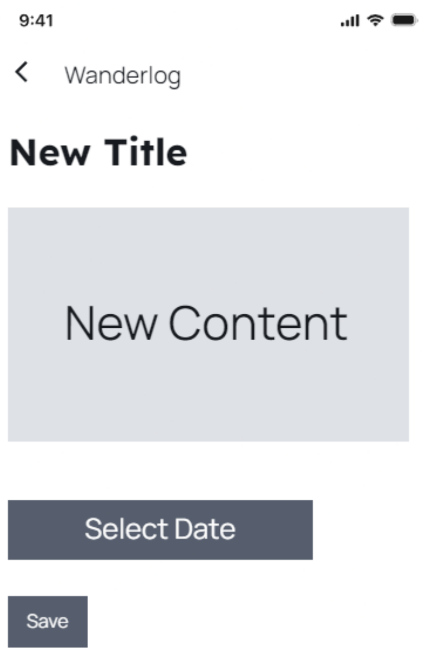
High-fidelity Wireframing Design for EntryActivity



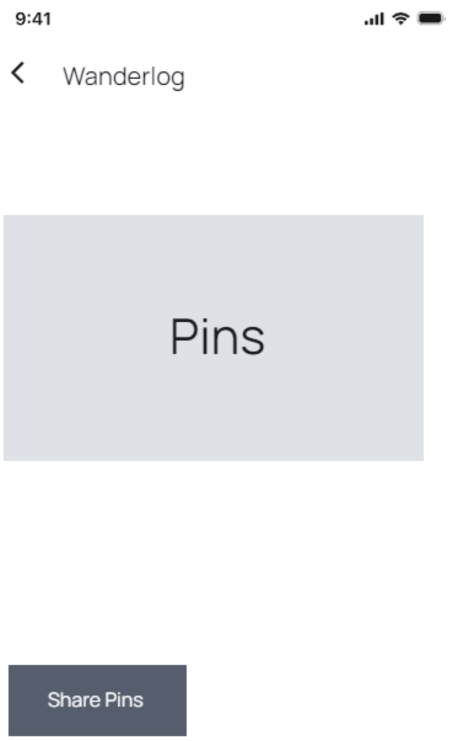
High-fidelity Wireframing Design for EntryDetailActivity



High-fidelity Wireframing Design for EditEntryActivity



High-fidelity Wireframing Design for PinsActivity



Illustrating Core Functions of the Journal App

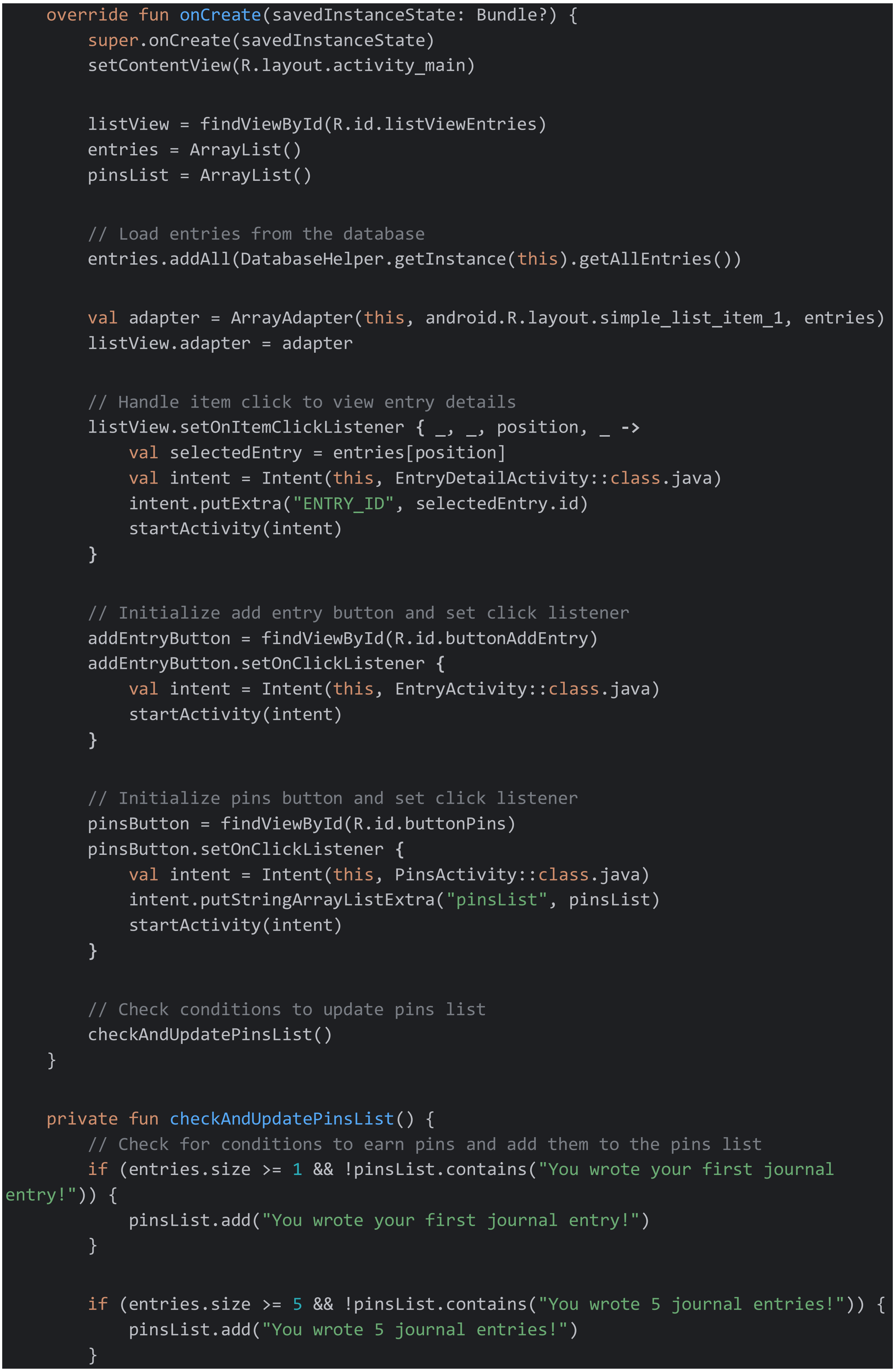
The journal app's core functions include navigation between screens, handling user interactions, setting up data structures, and accessing external services. Below are relevant extracts of the source code with detailed explanations for each core function.

1. Navigation Between Screens

Navigation in the app is primarily managed using Intents. Here’s how navigation is implemented between different activities.

MainActivity.kt

```
class MainActivity : AppCompatActivity() {
    private lateinit var listView: ListView
    private lateinit var entries: ArrayList<JournalEntry>
    private lateinit var addEntryButton: Button
    private lateinit var pinsButton: Button
    private lateinit var pinsList: ArrayList<String>
```



```
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        listView = findViewById(R.id.listViewEntries)
        entries = ArrayList()
        pinsList = ArrayList()

        // Load entries from the database
        entries.addAll(DatabaseHelper.getInstance(this).getAllEntries())

        val adapter = ArrayAdapter(this, android.R.layout.simple_list_item_1, entries)
        listView.adapter = adapter

        // Handle item click to view entry details
        listView.setOnItemClickListener { _, _, position, _ ->
            val selectedEntry = entries[position]
            val intent = Intent(this, EntryDetailActivity::class.java)
            intent.putExtra("ENTRY_ID", selectedEntry.id)
            startActivity(intent)
        }

        // Initialize add entry button and set click listener
        addEntryButton = findViewById(R.id.buttonAddEntry)
        addEntryButton.setOnClickListener {
            val intent = Intent(this, EntryActivity::class.java)
            startActivity(intent)
        }

        // Initialize pins button and set click listener
        pinsButton = findViewById(R.id.buttonPins)
        pinsButton.setOnClickListener {
            val intent = Intent(this, PinsActivity::class.java)
            intent.putStringArrayListExtra("pinsList", pinsList)
            startActivity(intent)
        }

        // Check conditions to update pins list
        checkAndUpdatePinsList()
    }

    private fun checkAndUpdatePinsList() {
        // Check for conditions to earn pins and add them to the pins list
        if (entries.size >= 1 && !pinsList.contains("You wrote your first journal
entry!")) {
            pinsList.add("You wrote your first journal entry!")
        }

        if (entries.size >= 5 && !pinsList.contains("You wrote 5 journal entries!")) {
            pinsList.add("You wrote 5 journal entries!")
        }
        // Add more conditions to earn other pins as needed

        // Ensure the pins list is up to date before starting PinsActivity
        pinsButton.setOnClickListener {
            val intent = Intent(this, PinsActivity::class.java)
            intent.putStringArrayListExtra("pinsList", pinsList)
            startActivity(intent)
        }
    }
```

Explanation:

Intents: Used to navigate between activities (EntryActivity, EntryDetailActivity, and PinsActivity).

onCreate: Sets up the initial state, including ListView, buttons, and loading data from the database.

onResume: Ensures the data is refreshed when returning to MainActivity from another activity.

2. Handling User Interactions

User interactions are primarily managed through button clicks and ListView item selections.

EntryActivity.kt

```
class EntryActivity : BaseActivity() {

    private lateinit var titleEditText: EditText
    private lateinit var contentEditText: EditText
    private lateinit var saveButton: Button
    private lateinit var dateEditText: EditText

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_entry)

        titleEditText = findViewById(R.id.editTextTitle)
        contentEditText = findViewById(R.id.editTextContent)
        saveButton = findViewById(R.id.buttonSave)
        dateEditText = findViewById(R.id.editTextDate)

        dateEditText.setOnClickListener {
            showDatePickerDialog()
        }

        saveButton.setOnClickListener {
            val title = titleEditText.text.toString().trim()
            val content = contentEditText.text.toString().trim()
            val date = dateEditText.text.toString().trim() // Get the selected date

            if (title.isNotEmpty() && content.isNotEmpty() && date.isNotEmpty()) {
                val entry = JournalEntry(title = title, content = content, date =
date) // Pass the selected date
                val id = DatabaseHelper.getInstance(this).addEntry(entry)
                entry.id = id.toInt()
                finish()
            } else {
                // Show error message if title, content, or date is empty
                Toast.makeText(this, "Title, content, and date cannot be empty",
Toast.LENGTH_SHORT).show()
            }
        }
    }
```

Explanation:

Button Click Listeners: buttonSelectDate shows a date picker dialog, while buttonSave saves the journal entry to the database.

Database Interaction: Adds a new entry to the database using DatabaseHelper.

3. Setting Up Data Structures

The app uses a custom data class for journal entries and a singleton database helper class.

JournalEntry.kt

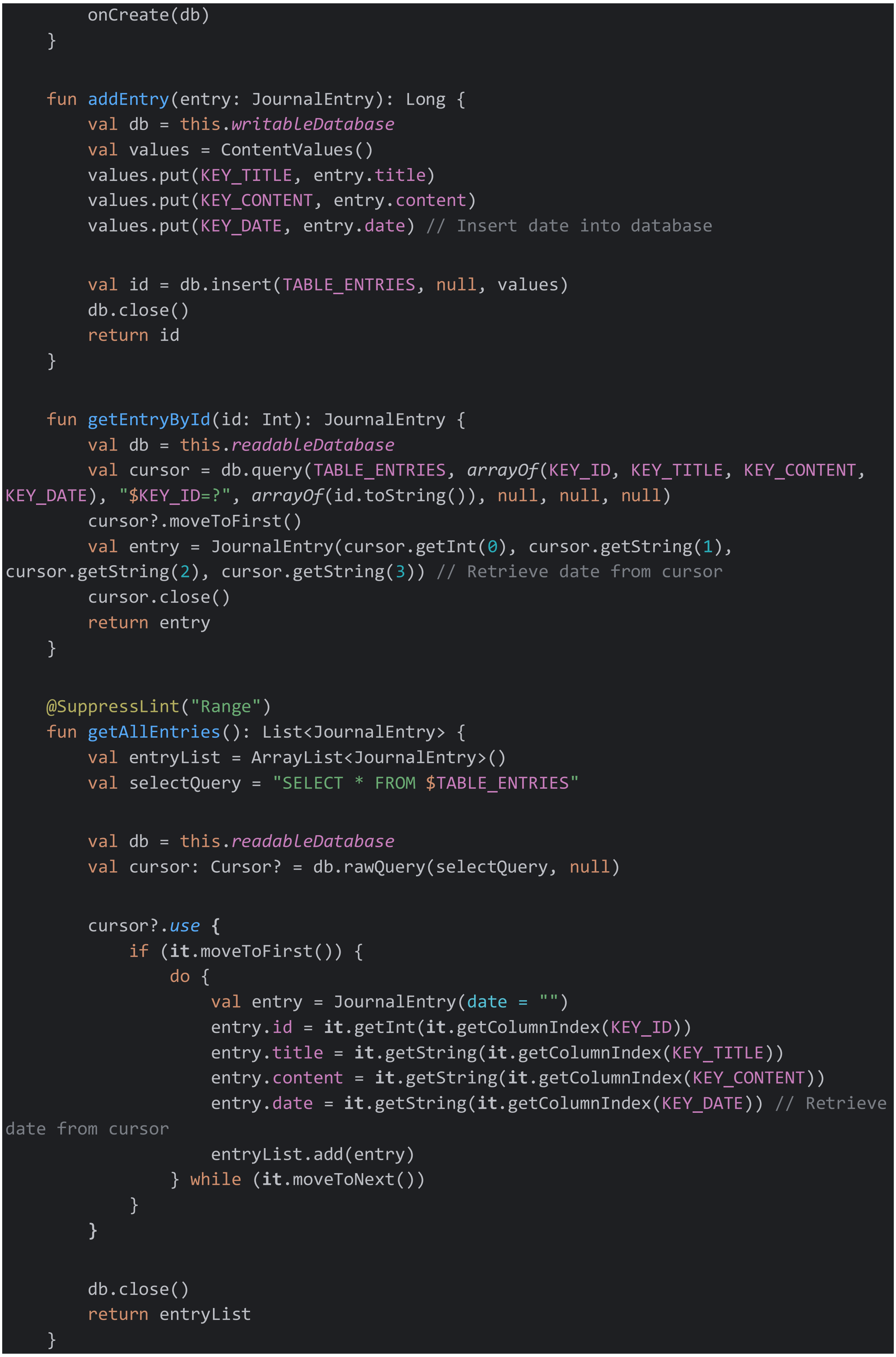
```
package com.example.wanderlog

data class JournalEntry(
    var id: Int = -1,
    var title: String = "",
    var content: String = "",
    var date: String
) {
    override fun toString(): String {
        return title
    }
}
```

Explanation:

Data Class: JournalEntry encapsulates the data for a journal entry with properties for id, title, content, and date.

DatabaseHelper.kt

```
class DatabaseHelper public constructor(context: Context) : SQLiteOpenHelper(context, DATABASE_NAME, null, DATABASE_VERSION) {

    companion object {
        private const val DATABASE_VERSION = 2 // Increment version to 2
        private const val DATABASE_NAME = "WanderlogDB"
        private const val TABLE_ENTRIES = "entries"

        // Column names including the date
        private const val KEY_ID = "id"
        private const val KEY_TITLE = "title"
        private const val KEY_CONTENT = "content"
        private const val KEY_DATE = "date" // New column for date

        private var instance: DatabaseHelper? = null

        @Synchronized
        fun getInstance(context: Context): DatabaseHelper {
            if (instance == null) {
                instance = DatabaseHelper(context.applicationContext)
            }
            return instance!!
        }
    }

    override fun onCreate(db: SQLiteDatabase) {
        val createEntriesTable = ("CREATE TABLE $TABLE_ENTRIES($KEY_ID INTEGER PRIMARY KEY, $KEY_TITLE TEXT, $KEY_CONTENT TEXT, $KEY_DATE TEXT)") // Add date column
        db.execSQL(createEntriesTable)
    }

    override fun onUpgrade(db: SQLiteDatabase, oldVersion: Int, newVersion: Int) {
        db.execSQL("DROP TABLE IF EXISTS $TABLE_ENTRIES")
```



```kotlin
        onCreate(db)
    }

    fun addEntry(entry: JournalEntry): Long {
        val db = this.writableDatabase
        val values = ContentValues()
        values.put(KEY_TITLE, entry.title)
        values.put(KEY_CONTENT, entry.content)
        values.put(KEY_DATE, entry.date) // Insert date into database

        val id = db.insert(TABLE_ENTRIES, null, values)
        db.close()
        return id
    }

    fun getEntryById(id: Int): JournalEntry {
        val db = this.readableDatabase
        val cursor = db.query(TABLE_ENTRIES, arrayOf(KEY_ID, KEY_TITLE, KEY_CONTENT, KEY_DATE), "$KEY_ID=?", arrayOf(id.toString()), null, null, null)
        cursor?.moveToFirst()
        val entry = JournalEntry(cursor.getInt(0), cursor.getString(1), cursor.getString(2), cursor.getString(3)) // Retrieve date from cursor
        cursor.close()
        return entry
    }

    @SuppressLint("Range")
    fun getAllEntries(): List<JournalEntry> {
        val entryList = ArrayList<JournalEntry>()
        val selectQuery = "SELECT * FROM $TABLE_ENTRIES"

        val db = this.readableDatabase
        val cursor: Cursor? = db.rawQuery(selectQuery, null)

        cursor?.use {
            if (it.moveToFirst()) {
                do {
                    val entry = JournalEntry(date = "")
                    entry.id = it.getInt(it.getColumnIndex(KEY_ID))
                    entry.title = it.getString(it.getColumnIndex(KEY_TITLE))
                    entry.content = it.getString(it.getColumnIndex(KEY_CONTENT))
                    entry.date = it.getString(it.getColumnIndex(KEY_DATE)) // Retrieve date from cursor
                    entryList.add(entry)
                } while (it.moveToNext())
            }
        }

        db.close()
        return entryList
    }
```

Explanation:

Singleton Pattern: Ensures a single instance of DatabaseHelper is used throughout the app.

Database Operations: Methods for creating the table, adding entries, and retrieving all entries.

4. Accessing External Services

In the current implementation, the app uses implicit intents to share pins on social media platforms.

PinsActivity.java

```
private fun sharePins() {
    // Share pins on Instagram
    val instagramIntent = Intent(Intent.ACTION_SEND)
    instagramIntent.type = "text/plain"
    instagramIntent.putExtra(Intent.EXTRA_TEXT, "Check out my pins on wanderlog.com")
    instagramIntent.setPackage("com.instagram.android")

    // Share pins on Twitter
    val twitterIntent = Intent(Intent.ACTION_SEND)
    twitterIntent.type = "text/plain"
    twitterIntent.putExtra(Intent.EXTRA_TEXT, "Check out my pins on wanderlog.com")
    twitterIntent.setPackage("com.twitter.android")

    // Share pins on Facebook
    val facebookIntent = Intent(Intent.ACTION_SEND)
    facebookIntent.type = "text/plain"
    facebookIntent.putExtra(Intent.EXTRA_TEXT, "Check out my pins on wanderlog.com")
    facebookIntent.setPackage("com.facebook.katana")

    // Create a chooser dialog to allow the user to select the sharing app
    val chooserIntent = Intent.createChooser(Intent(), "Share via")

    // Add the sharing intents to the chooser dialog
    chooserIntent.putExtra(Intent.EXTRA_INITIAL_INTENTS, arrayOf(instagramIntent,
twitterIntent, facebookIntent))

    // Start the chooser dialog
    startActivity(chooserIntent)
}
```

Explanation:

Implicit Intent: Used to share the pins via other apps installed on the device.

Button Click Listener: Checks if there are pins to share and initiates the sharing process using an implicit intent.

Evaluation the Journal App

<u>Fulfillment of Targeted Functionality</u>

The developed journal application proves to be reliable in meeting the distinct functionalities as listed during the project requirements. Key points of functionality include:

Add, Update, View, and Delete Journal Entries: Journals can be easily created and blog posts can be added with titles, content, and date. Preexisting entries can be updated, it is possible to take a deep dive into them, and they can be deleted. The developing is robust, with no stitching between the add, edit, and detail views.

Overview and Detailed View of Entries: This application offers a ListView, which gives a summary of all journal entries in an organized manner. When a entry is clicked the detail page opens up, displaying the full content of the entry.

Multiple Android Activities: The app performs several tasks (MainActivity, EntryActivity, and EntryDetailActivity) to offer user better and simplified treatment.

Unit Testing: The app has passed10 unit tests all right, which verifies the main functions are operating as required. Here is where the set of activities including adding, updating, deleting entries, and transitioning between screens are covered.

Material Design and Android App Quality Guidelines: The app follows material design guidelines, and this is carried through the interface to ensure a pleasant user experience for its audience. Comments and libraries used are well organized, which makes the code understandable and error-free.

Source Code Documentation and GitHub Repository: All the relative source code is well documented, therefore making it possible for others to follow and maintain. All code, resources, and config files are uploaded to a GitHub repository, where they can be used by other developers.

Areas for Improvement

While the app meets its core requirements, there are several enhancements that could improve its functionality and user experience:

Incorporating Media

Provide option to add notes, pictures and audio notes to journals. It ensures thus the app's interactivity and the publishers' capability to produce more informative subjects.

Notifications

Introduce alerts to inform when new transactions should be entered or old records are revisited. Consequently, this will permit to keep up journaling routine, be it daily or weekly. The ability to choose notification types and frequency would provide users with better control over the reminders that they get.

Cloud Sync and Backup

Incorporate the technology of cloud synchronization so that the users are able to access their entries from various devices. Give the user the ability to set self-backups to the protect data.

Lessons Learned

Importance of Planning and Design:

Detailed planning and development design which is responsible for a usable and maintainable app are entirely important. The prototypes and design documents of software served as effective guides to eventually smoothen the development.

User-Centered Design:

The key to an app that is successful is listening and learning about what users want and like, and then basing the app's design on those needs. The lesson applied was to the app being easy to use and visually appealing had been the primary factor.

Modular Development:

We split the app up into smaller, separated modules which allowed us to control the development process and the testing and debugging became easier.

Testing and Quality Assurance:

With the use of unit tests right from the development stage, all the bugs and errors have been promptly addressed to prevent the occurrence of anymore and have an application that is reliable and robust.

Documentation and Code Quality:

The documentation has been maintained well and coding standard followed well, thus code readability and maintainability has improved. Jamming in the future developers to contribute to the project is now easier.

Adaptability and Continuous Improvement:

The key to ongoing improvement lies not only in being willing to accept feedback but also in making fine-tuning of the design and functionality an ongoing exercise. The proposed measures serve to pinpoint the specific concerns to be addressed in an upgrade plan.

Conclusion

The journal app project has successfully achieved core functionality level, provisioning a robust self-management system that allows users to collect and store their memories. For the primary features, such as adding, modifying, presenting, and deleting diary entries, along with the overview and detailed review of entries, implementation has been demonstrated compellingly. Material design principles are followed and Android app quality guidelines are adhered to by the app, which in turn gives the users an experience, which is both intuitive and consistent. Moreover, the outcome of this 10 unit test serve to demonstrate how resilient and robust the aspects of the core properties are.

However, nothing is perfect and we can always try to be better. Some future features like adding media (sticker, images, notifications) as well as notifications can be developed to improve the user experience. Also, better interface can be done to make the user interface more attractive. Furthermore, data management options such as tag and category can be added to the application to provide more chances for the users. The cloud storage integration, regular and automatic backups, and the social media sharing functions would further improve the app functionality and the level of involvement of its users.

Along the way, there were several lessons, some of these became important. A great deal of value was attached to detailed planning and designing, user driven development, modular development, extensive testing, strict code quality, and the virtue of continuous improvement. Such learning will act as my compass in the new projects to come, enabling me to progress towards greater innovativeness and success.

Generally, the app has created a solid basis for the next release of the digital journaling interface, which will be interesting and versatile. The implications of such changes can help the service to develop into the more sophisticated instrument where users could collect, order and convey their memories and impressions. While this project may be my first time in software development, the lessons and skills learnt from this phase will always be a driving factor in the future projects I embark on.

Bibliography and Resources:

Google Developers (2022). Design for Android. https://developer.android.com/design.

t2informatik (2022): Wireframing. https://t2informatik.de/en/smartpedia/wireframing/?noredirect=en-US.

GitHub (2022). GitHub Docs. Repositories. https://docs.github.com/en/repositories.

Kotlinlang - https://kotlinlang.org/docs/coding-conventions.html#source-file-names

Kotlinlang - https://kotlinlang.org/docs/list-operations.html

Gradle - https://docs.gradle.org/current/samples/sample_building_groovy_applications.html

Gradle - https://docs.gradle.org/current/userguide/gradle_wrapper_basics.html

Kotlin Training - https://kotlinlang.org/education/

JUnit Test Training - https://www.coursera.org/learn/j-unit-testing

GitHub Repository Link:
https://github.com/zdemirgm/20240517_Zehra_Demir_92130119_DLBCSEMSE02.git