Inline Skate Academy

For inline skaters, who need to learn on the go.
Inline Skate Academy is an android mobile app for learning inline skating
that supports categorized lessons with helpful videos and instructions.
Unlike all competitors it can store user notes.

1. **App features**: inline skate learning content, local notes
2. **Market Proposition**: Market-Driven: Helpful videos to guide through the inline skating learning process with the ability to add notes. So users can see their notes for the lessons without having to switch to a notes app.

> The project aims to create an exploratory Android app prototype for Prof. Kurpjuweit's semester work. Additionally, testing with potential users will provide feedback and reveal hidden preferences, potentially expanding functionality.

About me

- Name: Nhat-Lam Luong
  - Matrikel number: 675523
  - email: inf3381@hs-worms.de
- Occupation: IT Student in HS-Worms
- Hobbies: Passion for Inline Skating and Information Technology

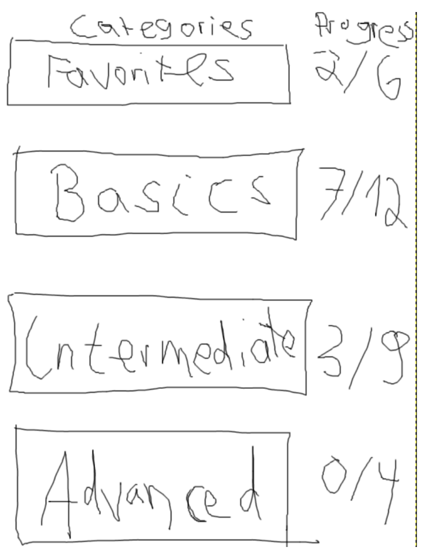
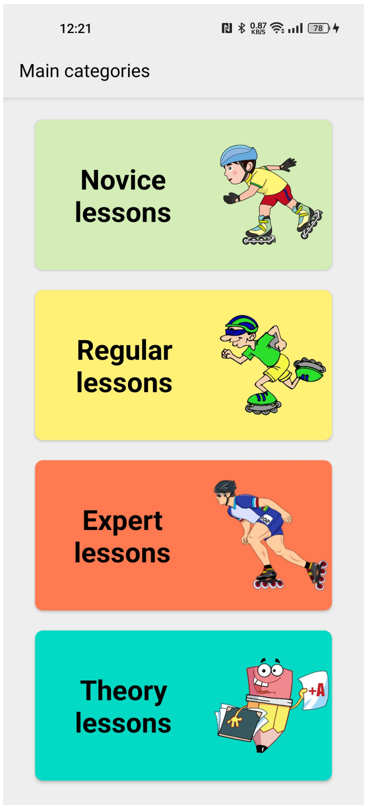
---

Project Development

Home Screen

The Home Screen or `MainActivity.class` is accessed directly after the initial app launch and its primary purpose is to provide an overview of all available categories [HS-1].

ID	User Story
HS-1	A user can have an overview of all categories **[DONE]**
HS-2	A user can see how far their progress for the categories are **[WIP]**
HS-3	A user can access their favorite lessons **[WIP]**

Wireframe (initial plan)	Status quo
 Categories Progress Favorites 2/6 Basics 7/12 Intermediate 3/9 Advanced 0/4	 12:21 Main categories Novice lessons Regular lessons Expert lessons Theory lessons

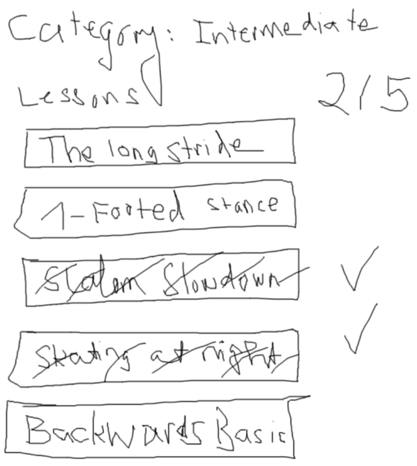
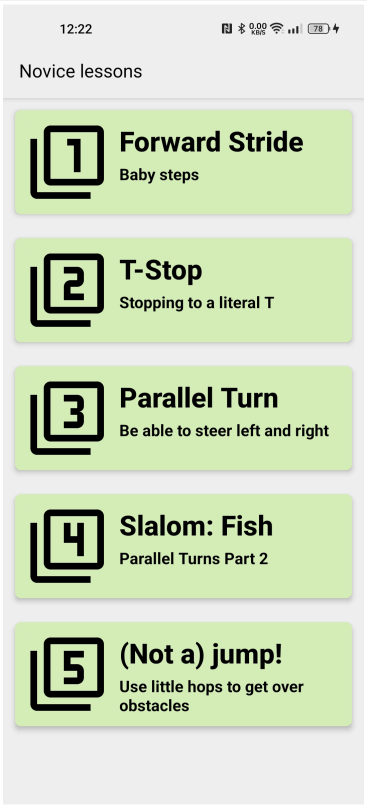
Code Reference

```
/* This is the entry point and home screen of the app, where it shows different categories of
lessons. */
public class MainActivity extends AppCompatActivity {

    /* These CardViews represent the lesson categories */
    private void findViews() {
        noviceCardView = findViewById(R.id.novice_cardview);
        regularCardView = findViewById(R.id.regular_cardview);
        expertCardView = findViewById(R.id.expert_cardview);
        theoryCardView = findViewById(R.id.theory_cardview);
    }

    /* Sets up click listeners for each CardView, so the app switches to the matching category
activity. */
    private void setupClickListeners() {
        noviceCardView.setOnClickListener(view -> switchToCategory(NoviceCategoryActivity.class));
        regularCardView.setOnClickListener(view -> switchToCategory(RegularCategoryActivity.class));
        expertCardView.setOnClickListener(view -> switchToCategory(ExpertCategoryActivity.class));
        theoryCardView.setOnClickListener(view -> switchToCategory(TheoryCategoryActivity.class));
    }

    private void switchToCategory(Class<?> categoryActivityClass) {
        Intent intent = new Intent(this, categoryActivityClass);
        startActivity(intent);
    }
}
```

Category Screen

After the user has clicked on a category `CardView` in the home screen, a new category screen like `TheoryCategoryActivity.class` will be opened and its primary purpose is to provide an overview of its available lessons [CTG-1].

ID	User Story
CTG-1	A user can have an overview of all lessons **[DONE]**
CTG-2	A user can see which lessons have been marked as done. **[WIP]**

Wireframe	Status quo
	

12:22

Novice lessons

1 **Forward Stride**
Baby steps

2 **T-Stop**
Stopping to a literal T

3 **Parallel Turn**
Be able to steer left and right

4 **Slalom: Fish**
Parallel Turns Part 2

5 **(Not a) jump!**
Use little hops to get over obstacles

Code Reference

Since the category screens have the same layout and only differentiate from their description and list of lessons, an abstract class has been implemented to minimize code duplication.

For instance, the following are two code snippets that illustrate the abstract relationship between a parent and a child. Afterward, there are additional code snippets to demonstrate the usage of the `RecyclerView`.

```
/* BaseCategoryActivity is an abstract parent class, where it has the common UI components for
displaying the category lessons */
public abstract class BaseCategoryActivity extends AppCompatActivity {

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_base_category);

        setupTitle();
        /* The RecyclerView for displaying lessons */
        /* It is advised to use Recyclers, so the environment of planet earth can be prolonged */
        setupRecyclerView();
    }

    private void setupTitle() {
        setTitle(getCategoryTitle());
    }

    /* Sets up the RecyclerView to display lessons by retrieving the lessons array and creating an
OnItemClickListener to handle lesson clicks. */

    private void setupRecyclerView() {
        RecyclerView recyclerView = findViewById(R.id.recycler_view_base_category);

        Lesson[] lessons = getLessons();
        LessonAdapter.OnItemClickListener listener = lesson -> {
            Intent intent = getIntentForLesson(lesson);
            if (intent != null) {
                startActivity(intent);
            }
        };
        LessonAdapter adapter = new LessonAdapter(lessons);
        adapter.setOnItemClickListener(listener);
        recyclerView.setAdapter(adapter);
    }

        /* Those are the abstract methods for the categories, since not every category has the same
title or lessons */
    protected abstract int getCategoryTitle();
    protected abstract Lesson[] getLessons();
    /* For launching the matching activity related to the specific lesson. */
    protected abstract Intent getIntentForLesson(Lesson lesson);
}
```

```
/* NoviceCategoryActivity is a child class of BaseCategoryActivity and it provides the list of
novice category lessons. */

public class NoviceCategoryActivity extends BaseCategoryActivity {

    @Override
    protected int getCategoryTitle() {
        return R.string.novice_title;
    }

    /* Provides an array of Lesson objects representing the novice category lessons, each with its
details */
    @Override
    protected Lesson[] getLessons() {
        return new Lesson[] {
                new Lesson(getString(R.string.novice_lesson_1_title),
getString(R.string.novice_lesson_1_description), R.drawable.baseline_filter_1_24,
ContextCompat.getColor(this, R.color.novice), ""),
                new Lesson(getString(R.string.novice_lesson_2_title),
getString(R.string.novice_lesson_2_description), R.drawable.baseline_filter_2_24,
ContextCompat.getColor(this, R.color.novice), ""),
                new Lesson(getString(R.string.novice_lesson_3_title),
getString(R.string.novice_lesson_3_description), R.drawable.baseline_filter_3_24,
ContextCompat.getColor(this, R.color.novice), ""),
                new Lesson(getString(R.string.novice_lesson_4_title),
getString(R.string.novice_lesson_4_description), R.drawable.baseline_filter_4_24,
ContextCompat.getColor(this, R.color.novice), ""),
                new Lesson(getString(R.string.novice_lesson_5_title),
getString(R.string.novice_lesson_5_description), R.drawable.baseline_filter_5_24,
ContextCompat.getColor(this, R.color.novice), "")
        };
    }

    /* Matches each lesson's title to the corresponding activity class. */
    @Override
    protected Intent getIntentForLesson(Lesson lesson) {
        Intent intent = null;
        if (getString(R.string.novice_lesson_1_title).equals(lesson.getTitle())) {
            intent = new Intent(this, NoviceLesson1Activity.class);
        } else if (getString(R.string.novice_lesson_2_title).equals(lesson.getTitle())) {
            intent = new Intent(this, NoviceLesson2Activity.class);
```

```java
        } else if (getString(R.string.novice_lesson_3_title).equals(lesson.getTitle())) {
            intent = new Intent(this, NoviceLesson3Activity.class);
        } else if (getString(R.string.novice_lesson_4_title).equals(lesson.getTitle())) {
            intent = new Intent(this, NoviceLesson4Activity.class);
        } else if (getString(R.string.novice_lesson_5_title).equals(lesson.getTitle())) {
            intent = new Intent(this, NoviceLesson5Activity.class);
        }
        return intent;
    }
}
```

RecyclerView

To dive deeper for the `RecyclerView` please refer to the following code sections and its included comments:

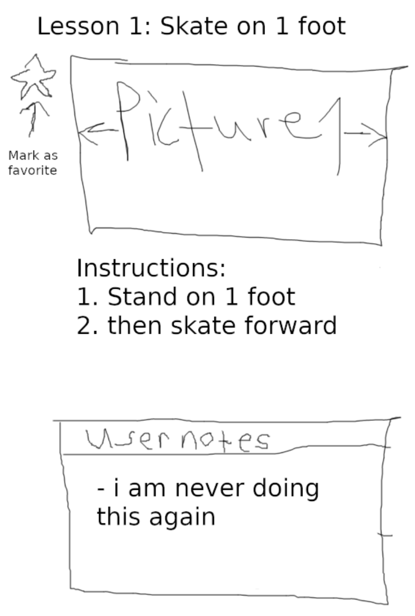
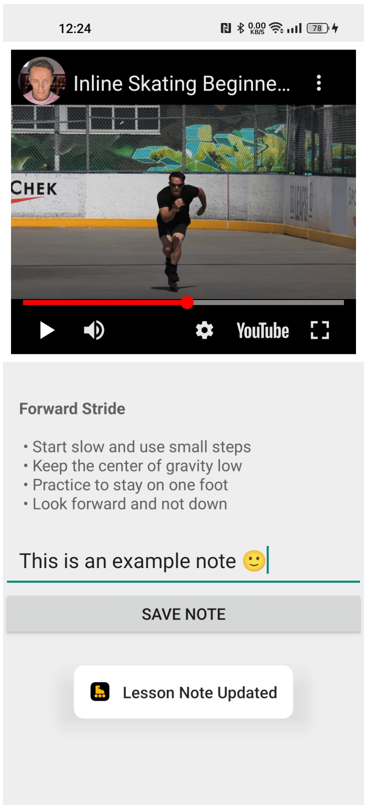
```java
/* Represents a lesson item in the RecyclerView. */
public class Lesson {

    final String title;
    final String description;
    final int image;
    final int backgroundColor;
    final String webLink;

    public Lesson(String title, String description, int image, int backgroundColor, String webLink)
{
        this.title = title;
        this.description = description;
        this.image = image;
        this.backgroundColor = backgroundColor;
        this.webLink = webLink;
    }

    public String getTitle() {
        return title;
    }

    public String getWebLink() {
        return webLink;
    }
}
```

```java
/* This class manages the display of lessons for the RecyclerView. */
public class LessonAdapter extends RecyclerView.Adapter<LessonAdapter.LessonViewHolder> {

    /* Initializes the adapter with an array of lessons. */
    public LessonAdapter(Lesson[] lessons) {
        this.lessons = lessons;
    }

    public void setOnItemClickListener(OnItemClickListener listener) {
        this.listener = listener;
    }

    /* Inflate the item layout to create a new ViewHolder for each lesson item. */
    @NonNull
    @Override
    public LessonViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        View view = LayoutInflater.from(parent.getContext())
                .inflate(R.layout.item_lesson, parent, false);
        return new LessonViewHolder(view);
    }
    /* Binds data to the ViewHolder with matching lesson content */
    @Override
    public void onBindViewHolder(@NonNull LessonViewHolder holder, int position) {
        holder.bindViewHolder(lessons[position]);
    }

    @Override
    public int getItemCount() {
        return lessons.length;
    }

    public interface OnItemClickListener {
        void onItemClick(Lesson lesson);
    }

    /* Responsible for holding and displaying individual lesson items. */
    class LessonViewHolder extends RecyclerView.ViewHolder {

        /* Constructor to initialize views within the ViewHolder. */
        LessonViewHolder(@NonNull View itemView) {
            super(itemView);
            lessonImage = itemView.findViewById(R.id.lesson_image);
            lessonTitle = itemView.findViewById(R.id.lesson_title);
            lessonDescription = itemView.findViewById(R.id.lesson_description);
            lessonBackgroundColor = itemView.findViewById(R.id.card_view_lesson);
            itemView.setOnClickListener(v -> onItemClick());
        }

        /* Fill the ViewHolder's views with data from the matching lesson. */
        void bindViewHolder(Lesson lesson) {
            lessonTitle.setText(lesson.title);
            lessonDescription.setText(lesson.description);
            lessonImage.setImageResource(lesson.image);
            lessonImage.setBackgroundColor(lesson.backgroundColor);
            lessonBackgroundColor.setCardBackgroundColor(lesson.backgroundColor);
        }

        void onItemClick() {
            int position = getAbsoluteAdapterPosition();
            /* Position and listener have to be valid */
            if (position != RecyclerView.NO_POSITION && listener != null) {
                listener.onItemClick(lessons[position]);
            }
        }
    }
}
```

Lesson Screen

After the user has clicked on a lesson item `RecyclerView` in the category screen, a new lesson screen like `ExpertLesson1Activity.class` will be opened and its primary purpose is to provide helpful instructions in form of media [LSS-1], note saving functionality [LSS-2] and mark the lesson as favorite [LSS-3].

ID	User Story
LSS-1	A user can find helpful media for the lesson **[DONE]**
LSS-2	A user can save notes **[DONE]**
LSS-3	A user can mark a lesson as favorite **[WIP]**

Wireframe	Status quo
 Lesson 1: Skate on 1 foot Mark as favorite Picture 1 Instructions: 1. Stand on 1 foot 2. then skate forward User notes - i am never doing this again	 Forward Stride • Start slow and use small steps • Keep the center of gravity low • Practice to stay on one foot • Look forward and not down This is an example note 🙂 SAVE NOTE Lesson Note Updated

Code Reference

Since the lesson screens have the same layout and only differentiate from their descriptions, media and notes, an abstract class has been implemented to minimize code duplication.

For instance, the following are two code snippets that illustrate the abstract relationship between a parent and a child. Afterward, there are additional code snippets to demonstrate the usage of the `RoomDatabase`.

```java
/* The abstract parent class BaseLessonActivity.
It provides common functionality for displaying lesson content and managing lesson notes. */

public abstract class BaseLessonActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_base_lesson);

        /* Unique identifier for each lesson note */
        NOTE_ID = getString(getNoteId());

        /* Initialize all the layout components */
        defineContentView();
        /* this class provides Room ORM to interact with the LessonNoteDatabase.class */
        databaseHelper = new DatabaseHelper(this);

        /* Loads the note at the beginning, so the user can see their notes directly */
        loadLessonNoteInBackground();
    }

    /* After loading the note from the database, it updates the UI to display it. */
    private void loadLessonNoteInBackground() {
        new Thread(() -> {
            LessonNote lessonNote = databaseHelper.loadLessonNoteBackground(NOTE_ID);
            handler.post(() -> displayLessonNote(lessonNote));
        }).start();
    }

    private void displayLessonNote(LessonNote lessonNote) {
        if (lessonNote != null) {
            editLessonNote.setText(lessonNote.getNote());
        } else {
            UIHelper.showToast(this, getString(R.string.error_load_note));
        }
    }

    /* Initializes all components of the lesson layout. */
    @SuppressLint("SetJavaScriptEnabled")
    private void defineContentView() {
        // Init WebView for YouTube video
        WebView webView = findViewById(R.id.base_lesson_web_view);
        String video = getString(getYouTubeVideoEmbedCode());
        webView.loadData(video, "text/html", "utf-8");
        webView.getSettings().setJavaScriptEnabled(true);
        webView.setWebChromeClient(new WebChromeClient());

        // Init TextView for text tutorial
        TextView textView = findViewById(R.id.base_lesson_text_view);
        textView.setText(getTutorialText());

        // Init EditText for lesson note
        editLessonNote = findViewById(R.id.base_lesson_edit_text);

        // Init Button for updating lesson note
        Button updateLessonNote = findViewById(R.id.base_lesson_button);
        updateLessonNote.setOnClickListener(v -> updateLessonNoteBackground());
    }

    /* Updates the lesson note in the background by retrieving text from EditText and calls the
database helper to perform the update. */
    private void updateLessonNoteBackground() {
        String note = editLessonNote.getText().toString();
        databaseHelper.updateLessonNoteBackground(NOTE_ID, note);
        UIHelper.showToast(this, getString(R.string.note_updated));
    }
}
```

```
/* A child class of BaseLessonActivity, representing a specific lesson of the Expert category. */

public class ExpertLesson1Activity extends BaseLessonActivity {

    @Override
    public int getNoteId() {
        return R.string.expert_lesson_1_note_id;
    }
    @Override
    public int getYouTubeVideoEmbedCode() {
        return R.string.expert_lesson_1_yt_video;
    }
    @Override
    public int getTutorialText() {
        return R.string.expert_lesson_1_tutorial;
    }
}
```

RoomDatabase

To dive deeper for the `RoomDatabase` please refer to the following code sections and its included comments:

```
/* The LessonNote represents a note for a specific lesson. */
@Entity(tableName = "lesson_note")
public class LessonNote {

    @ColumnInfo(name = "id")
    @PrimaryKey
    @NonNull
    private final String id;

    @ColumnInfo(name = "note")
    private final String note;

    public LessonNote(@NonNull String id, String note) {
        this.id = id;
        this.note = note;
    }

    @NonNull
    public String getId() {
        return id;
    }

    public String getNote() {
        return note;
    }
}
```

```
/* This interface provides operations for interacting with the lesson_note table in the database. */
@Dao
public interface LessonNoteDAO {

    /* Inserts a new lesson note entry. */
    @Insert
    void addLessonNote(LessonNote lessonNote);

    /* Updates an existing lesson note */
    @Update
    void updateLessonNote(LessonNote lessonNote);

    /* Retrieve a specific lesson note by ID. */
    @Query("select * from lesson_note where id == :lessonNote_id")
    LessonNote getLessonNote(String lessonNote_id);
}
```

```
/* This is the DB itself and uses the DAO for operations. */
@Database(entities = {LessonNote.class}, version = 1, exportSchema = false)
@SuppressWarnings("all")
public abstract class LessonNoteDatabase extends RoomDatabase {

    /**
     * Retrieve the LessonNoteDAO for database operations related to lesson notes.
     * @return The LessonNoteDAO instance.
     */
    public abstract LessonNoteDAO getLessonNoteDAO();
}
```

```
/* This class provides Room ORM to interact with the LessonNoteDatabase. */
/* Honestly, I just made this class to make the BaseLessonActivity.class more compact */
public class DatabaseHelper {

    public DatabaseHelper(Context context) {
        RoomDatabase.Callback roomCallback = new RoomDatabase.Callback() {
            @Override
            public void onCreate(@NonNull SupportSQLiteDatabase db) {
                super.onCreate(db);
            }

            @Override
            public void onOpen(@NonNull SupportSQLiteDatabase db) {
                super.onOpen(db);
            }
        };

        // Build LessonNoteDatabase using Room databaseBuilder
        LessonNoteDatabase lessonNoteDatabase = Room.databaseBuilder(context,
LessonNoteDatabase.class, "lessonNoteDatabase")
                .addCallback(roomCallback)
                .build();

        // Initialize DAO and ExecutorService for background tasks
        lessonNoteDAO = lessonNoteDatabase.getLessonNoteDAO();
        executorService = Executors.newSingleThreadExecutor();
    }

    /* Updates the lesson note in the background to not interfere with the UI thread. */
    public void updateLessonNoteBackground(String noteId, String note) {
        LessonNote lessonNote = new LessonNote(noteId, note);
        executorService.execute(() -> {
            try {
                lessonNoteDAO.updateLessonNote(lessonNote);
            } catch (Exception e) {
                e.printStackTrace();
            }
        });
    }

    /* Loads the lesson note in the background to not interfere with the UI thread. */
    public LessonNote loadLessonNoteBackground(String noteId) {
        try {
            LessonNote lessonNote = lessonNoteDAO.getLessonNote(noteId);
            if (lessonNote == null) {
                lessonNote = new LessonNote(noteId, "");
                lessonNoteDAO.addLessonNote(lessonNote);
            }
            return lessonNote;
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }
}
```

Testing

For `unitTest` cases via `JUnit4.class`, please refer to the following:

- `LessonTest.class`
  - `testGetTitle()`
  - `testGetWebLink()`

For the instrumented `androidTest` cases via `AndroidJUnit4.class`, please refer to the following:

- `LessonNoteDAOTest.class`
  - `testAddAndGetLessonNote()`

- `LessonNoteDatabaseTest.class`
  - `testGetLessonNoteDAO()`

These primarily cover the Lesson and Database sections, but it is advised to add more unit tests that cover other areas as well, such as Category or Home Screen, in order to ensure 100% code coverage.

Time tracking

Date	Time (min)	TimeF(x)	Category	Description	User Story
06.04.23	60	60	Documentation	Define Idea	-
12.04.23	30	90	Home screen	Basic Buttons	HS
13.04.23	150	240	Home screen	Card views	HS
16.04.23	60	300	Misc	Splashscreen + App Icon	-
17.04.23	30	330	Misc	Light- and dark theme	-
17.05.23	180	510	Categories	RecyclerView	CTG
17.07.23	300	810	Lessons	Novice Lessons	LSS
21.07.23	30	840	Lessons	Regular Lessons	LSS
22.07.23	30	870	Lessons	Expert Lessons	LSS
04.08.23	90	960	Lessons	Theory Lessons	LSS
05.08.23	420	1380	Lessons	Room Database	LSS
06.08.23	220	1600	Misc	JUnit Tests	-
07.08.23	200	1800	Documentation	README.md	-

Afterword

I am really proud of this project because I turned my vision into reality with it. What makes it even more special is that this project combines my unique interest in inline skating with IT. Currently, there is no app like this in the small world of inline skating, so I saw this as a chance to take matters into my own hands and face the challenge head-on!

However, I encountered difficulties when my partner had to leave the project for personal reasons. This left me alone with a huge workload that I couldn't handle by myself. As a result, I had to simplify some aspects of my project and focus only on meeting the essential requirements, like Multiple Screens, RecyclerView, User Input and Storage, Offline Functionality with a local database, and Unit Tests.

Overall, this journey brought me happiness, taught me how to manage myself, and provided me with a lot of practice in coding.