Midterm

Building a Simple Task Management App in Android Using Kotlin

Prepared by Ibragimov Temirlan

Almaty, 27.10.2024

Table of contents

Executive summary

The goal of this assignment is to develop a simple task manager application for Android devices. To accomplish that, I used Android Studio IDE and Kotlin programming language. Other than that, I used the RecyclerView component to view tasks and used XML layout files to build the UI. As a result, I developed a simple app that helps you to keep track of your current tasks, if needed you can edit them or mark them as completed.

Introduction

Task managers were always relevant throughout history. Back in the days, people used to write their daily tasks to their notes, but now with the enhancement of mobile technologies, mobile apps that track the tasks are becoming more and more relevant. That's why, in this assignment I made a simple task manager app, which helps you to track your tasks, set their description and time.

My main motivation was to create something interesting and yet useful in daily life. I am really interested in the mobile development field, so I enjoyed doing this assignment a lot.

My application consists of creating, reading and editing. Also, I would like to state that the user interface (UI) is very simple, which may be helpful in creating a good user experience (UX).

Project objectives

This project involves developing a simple task management application for Android using Kotlin. The app will allow users to create, view, update, and delete tasks, providing a user-friendly interface. Also develop a functional mobile application using Android Studio with Kotlin, and to deepen understanding of the Android activity lifecycle.

Key Technologies and Libraries Used

- Android Studio (IDE)
- Kotlin
- RecyclerView (for displaying lists)

Overview of Android Development and Kotlin

Android development with Kotlin is considered one of the best practices in developing mobile products. As Android's official programming language, Kotlin offers a nice and easy syntax. Its seamless integration with Java allows developers to use both languages in the same project, taking advantage of Kotlin's powerful features, such as extension functions, and coroutines for asynchronous programming, along with built-in Java libraries. The Android development ecosystem powered by Android Studio provides a big set of tools, including emulators, debugging, and layout editors, to simplify app creation and testing. Android Development and Kotlin enable developers to build scalable, efficient, and user-friendly applications that keep pace with the fast evolving mobile industry.

Development Environment: Setting up Android Studio for Kotlin development is quite simple. After downloading and installing Android Studio, you can create a new project and choose Kotlin as your primary language. The IDE has various tools, such as layout editors, emulators, and debugging options, all of it is enabled in Kotlin, making it easier to build and test your applications.

Kotlin Overview: Kotlin brings several advantages to Android development. Its null safety feature helps prevent those frustrating null pointer exceptions by explicitly handling null values. Extension functions allow you to add new functionalities to existing classes without modifying their code, which is very nice. Coroutines simplify asynchronous programming, making tasks like network calls much more manageable. Compared to Java, Kotlin requires less boilerplate code, which leads to cleaner, more readable code. Plus, it's fully compatible with Java, allowing developers to scale to Kotlin gradually. As a result, Kotlin enhances the development experience and makes building modern Android apps more efficient.

Functions and Lambdas in Kotlin

- Function Implementation: I have a special class named TaskViewModel, which helps me to make CRUD operations:

```
fun addTaskItem(newTask: TaskItem)
{
    val list = taskItems.value
    list!!.add(newTask)
    taskItems.postValue(list)
}

fun updateTaskItem(id: UUID, name: String, desc: String, dueTime: LocalTime?)
{
    val list = taskItems.value
    val task = list!!.find { it.id == id }!!
    task.name = name
    task.desc = desc
    task.dueTime = dueTime
    taskItems.postValue(list)
}

fun setCompleted(taskItem: TaskItem)
{
    val list = taskItems.value
    val task = list!!.find { it.id == taskItem.id }!!
    if (task.completedDate == null)
        task.completedDate = LocalDate.now()
    taskItems.postValue(list)
}
```

```
fun deleteTaskItem(taskItem: TaskItem)
{
    val list = taskItems.value
    val task = list!!.find { it.id == taskItem.id }!!
    if (task != null) {
        list.remove(task)
        taskItems.postValue(list)
    }
}
```

- Using Lambdas

Here is an example of me using a lambda function in my TaskViewModel.kt file:

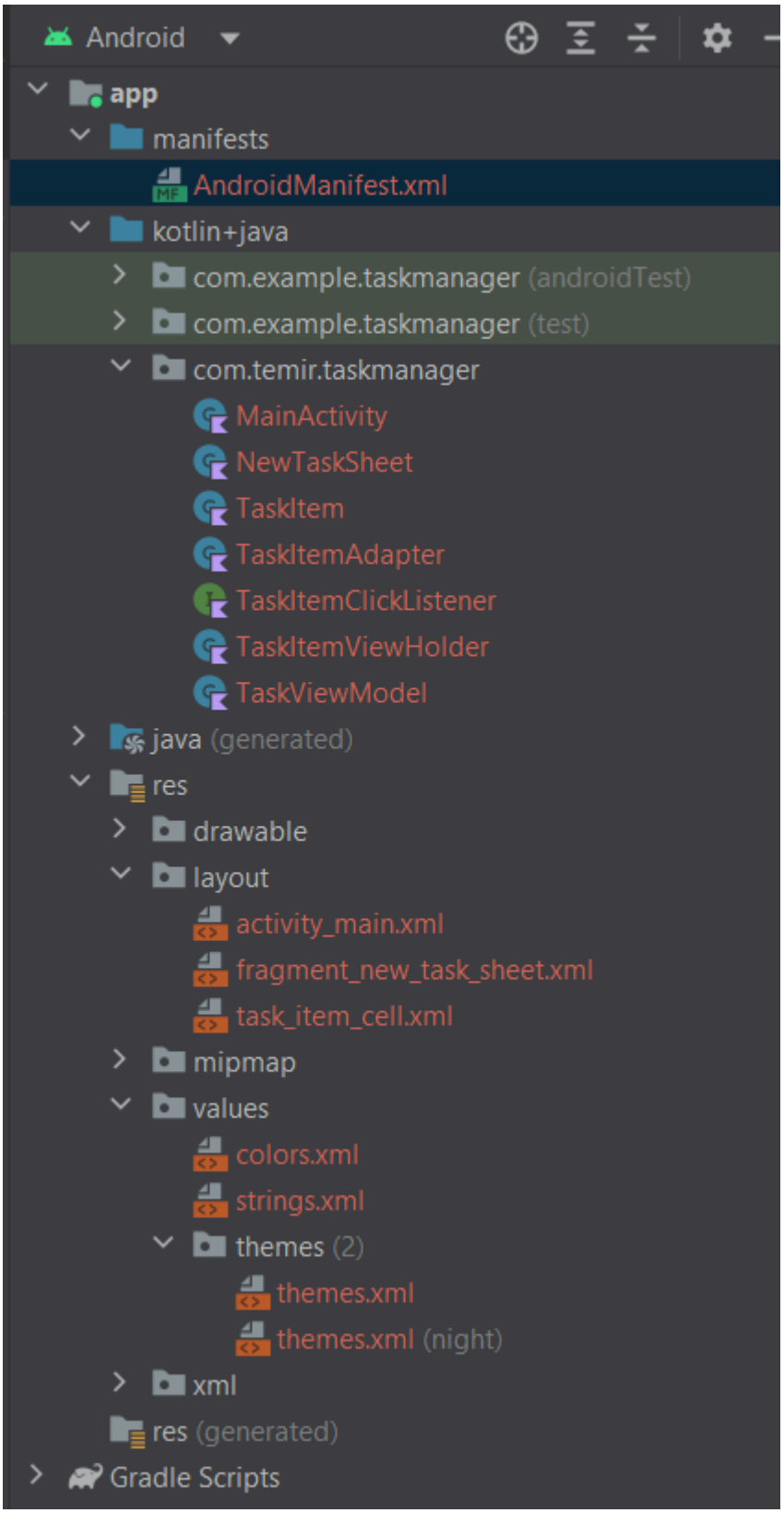
```
fun updateTaskItem(id: UUID, name: String, desc: String, dueTime: LocalTime?)
{
    val list = taskItems.value
    val task = list!!.find { it.id == id }!! // LAMBDA FUNCTION
    task.name = name
    task.desc = desc
    task.dueTime = dueTime
    taskItems.postValue(list)
}
```

Object-Oriented Programming in Kotlin

Classes and Data Classes

I created several classes with their own logic and responsibility.

- **MainActivity.kt** - runs the entire program and serves as the main class.
- **NewTaskSheet.kt** - is designed to create or edit a task using a user interface that slides up from the bottom of the screen.
- **TaskItem.kt** - a model of the tasks
- **TaskItemAdapter.kt** - RecyclerView adapter for displaying a list of TaskItem objects in a RecyclerView
- **TaskItemClickListener.kt** - interface to handle clicks
- **TaskItemViewHolder.kt** - class is responsible for binding the data of each TaskItem to the corresponding UI elements within a RecyclerView.
- **TaskViewModel.kt** - handles the data base operations (creating, deleting, updating, displaying).



Encapsulation

A very good example of encapsulation in my code is TaskItem class

```
class TaskItem(
    var name: String,
    var desc: String,
    var dueTime: LocalTime?,
    var completedDate: LocalDate?,
```

```
    var id: UUID = UUID.randomUUID()
)
{
    fun isCompleted() = completedDate != null
      fun imageResource(): Int = if(isCompleted()) R.drawable.checked_24 else
R.drawable.unchecked_24
     fun imageColor(context: Context): Int = if(isCompleted()) purple(context)
else black(context)

      private fun purple(context: Context) = ContextCompat.getColor(context,
R.color.purple_500)
      private fun black(context: Context) = ContextCompat.getColor(context,
R.color.black)
}
```

The name and desc properties are marked as private, so they cannot be accessed directly from outside the class. Public methods like getDetails(), updateName(), and updateDescription() provide access to modify the internal state.

Working with Collections in Kotlin

Collection Usage is implemented in TaskViewModel class. In this case data stored in a mutable list and then used throughout the lifecycle of a program as a data source.

Operations on Collections
Common operations on collections include filtering, sorting, and transforming data.

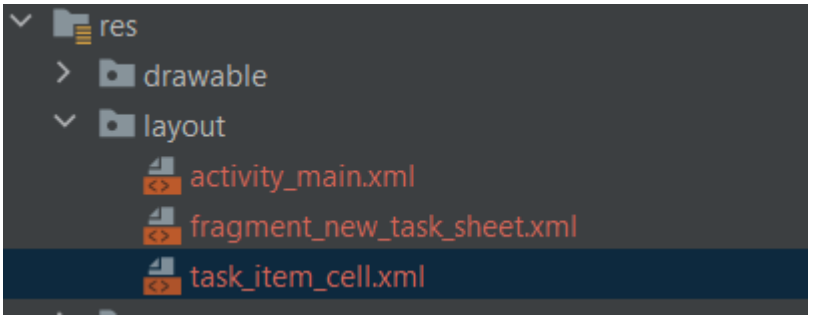
```
class TaskViewModel: ViewModel()
{
    var taskItems = MutableLiveData<MutableList<TaskItem>>()

    init {
        taskItems.value = mutableListOf()
    }

    fun addTaskItem(newTask: TaskItem)
    {
        val list = taskItems.value
        list!!.add(newTask)
        taskItems.postValue(list)
    }

        fun updateTaskItem(id: UUID, name: String, desc: String, dueTime:
LocalTime?)
    {
        val list = taskItems.value
        val task = list!!.find { it.id == id }!!
        task.name = name
        task.desc = desc
        task.dueTime = dueTime
        taskItems.postValue(list)
    }

    fun setCompleted(taskItem: TaskItem)
    {
        val list = taskItems.value
        val task = list!!.find { it.id == taskItem.id }!!
        if (task.completedDate == null)
            task.completedDate = LocalDate.now()
        taskItems.postValue(list)
    }
```

```
    fun deleteTaskItem(taskItem: TaskItem)
    {
        val list = taskItems.value
        val task = list!!.find { it.id == taskItem.id }!!
        if (task != null) {
            list.remove(task)
            taskItems.postValue(list)
        }
    }
}
```

Android Layout Design

In my application I used XML layouts to create UI components. I created three different XML layouts:

- **activity_main** - main layout file with RecyclerView
- **fragment_new_task_sheet** - layout of a popup with a form. Creates new task
- **task_item_cell** - a component layout which designs a single item of a task



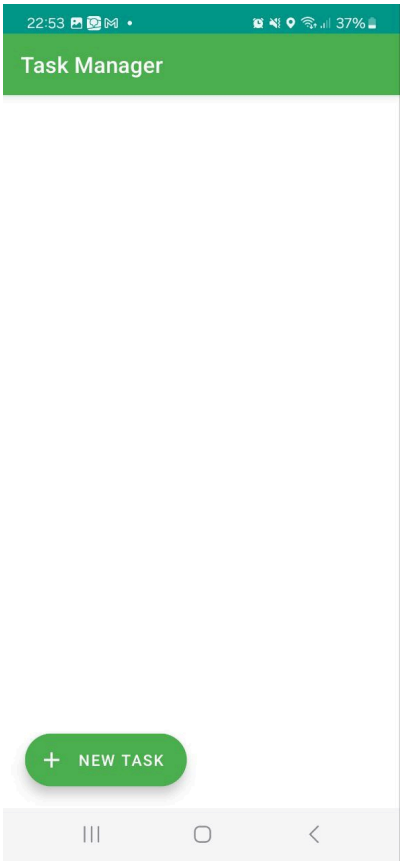
Choosing the right layout type is crucial for achieving an optimal user experience. Android offers several layout types. In my case, I used a combination of RelativeLayout and LinearLayout. RelativeLayout is used as a container, so that all of my objects would be relative and adaptive to any kind of android device. LinearLayout is used to display items in a row or in a column.

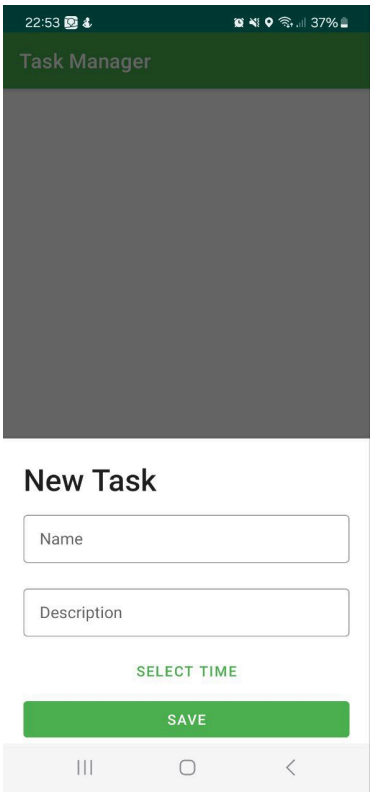
Activity and User Input Handling

The MainActivity class extends AppCompatActivity and implements the TaskItemClickListener interface, allowing it to respond to user interactions with task items. The newTaskButton's click listener opens a NewTaskSheet for adding new tasks. The activity implements methods from the TaskItemClickListener interface to handle editing, completing, and deleting tasks when the respective actions are triggered.

```
class MainActivity : AppCompatActivity(), TaskItemClickListener
{
    private lateinit var binding: ActivityMainBinding
    private lateinit var taskViewModel: TaskViewModel

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

```
        taskViewModel = ViewModelProvider(this).get(TaskViewModel::class.java)
        binding.newTaskButton.setOnClickListener {
            NewTaskSheet(null).show(supportFragmentManager, "newTaskTag")
        }
        setRecyclerView()
    }

    private fun setRecyclerView()
    {
        val mainActivity = this
        taskViewModel.taskItems.observe(this){
            binding.todoListRecyclerView.apply {
                layoutManager = LinearLayoutManager(applicationContext)
                adapter = TaskItemAdapter(it, mainActivity)
            }
        }
    }

    override fun editTaskItem(taskItem: TaskItem)
    {
        NewTaskSheet(taskItem).show(supportFragmentManager,"newTaskTag")
    }

    override fun completeTaskItem(taskItem: TaskItem)
    {
        taskViewModel.setCompleted(taskItem)
    }

    override fun deleteTaskItem(taskItem: TaskItem)
    {
        taskViewModel.deleteTaskItem(taskItem)
    }
}
```





Activity Lifecycle Management

Activity lifecycle management is essential in Android development because it ensures smooth application operation and state management during user interactions and system events. Basic lifecycle methods such as `**onCreate()**`, `**onStart()**`, `**onResume()**`, `**onPause()**`, `**onStop()**` and `**onDestroy()**` help manage resources, maintain and maintain the user experience. the state of the application during the transition. For example, overriding these methods in 'MainActivity' allows you to initialize UI components in '**onCreate()**', update data in **'onResume()**', pause current tasks in '**onPause()**' and clear resources in '. **onDestroy()**`, ensures that the application responds efficiently to user actions and system changes.

Fragments and Fragment Lifecycle

A Fragment represents a reusable portion of your app's UI. A fragment defines and manages its own layout, has its own lifecycle, and can handle its own input events. Fragments can't live on their own. They must be hosted by an activity or another fragment. The fragment's view hierarchy becomes part of, or attaches to, the host's view hierarchy.

In my app, fragments interact with the main activity primarily through the implementation of the TaskItemClickListener interface. When a user interacts with a fragment (for instance, by adding, editing, or completing a task), the fragment communicates these actions back to the main activity using the methods defined in the interface.

```
package com.temir.taskmanager


interface TaskItemClickListener

{
```

```
    fun editTaskItem(taskItem: TaskItem)

    fun completeTaskItem(taskItem: TaskItem)

    fun deleteTaskItem(taskItem: TaskItem)

}
```

MainActivity.kt code fragment:

```
override fun editTaskItem(taskItem: TaskItem)
{
    NewTaskSheet(taskItem).show(supportFragmentManager,"newTaskTag")
}

override fun completeTaskItem(taskItem: TaskItem)
{
    taskViewModel.setCompleted(taskItem)
}

override fun deleteTaskItem(taskItem: TaskItem)
{
    taskViewModel.deleteTaskItem(taskItem)
}
```

Conclusion

As a result, we received an application that fully complies with the requirements of this work, namely “Building a Simple Task Management App in Android Using Kotlin”. The application will allow users to create, view, update, and delete tasks.

The task manager app has turned out quite well by using modern Android technologies like Kotlin, which has made the code more readable and responsive. Fragments have added a nice touch of modularity to the app.

Looking ahead, it would be great to add features like deadline notifications, task prioritization, and a search function to enhance usability. Exploring cloud database options for data persistence and considering Jetpack Compose for a more modern UI design could also take the app to the next level.

References

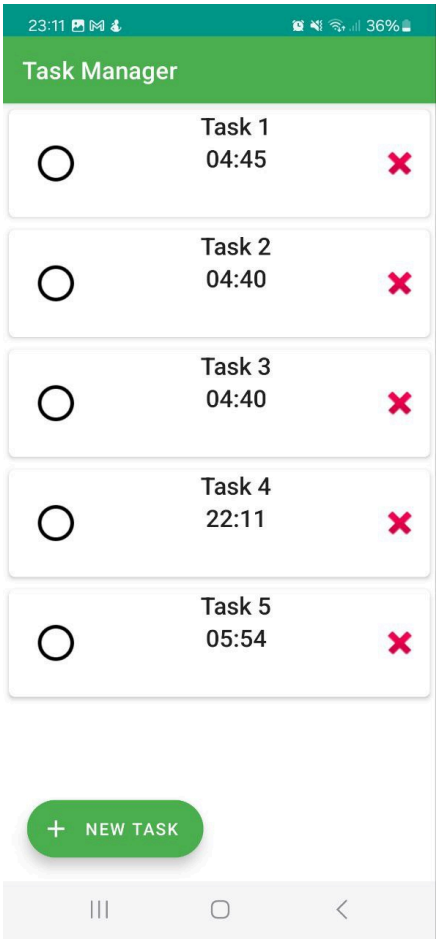
Links to Android Studio official documentation:

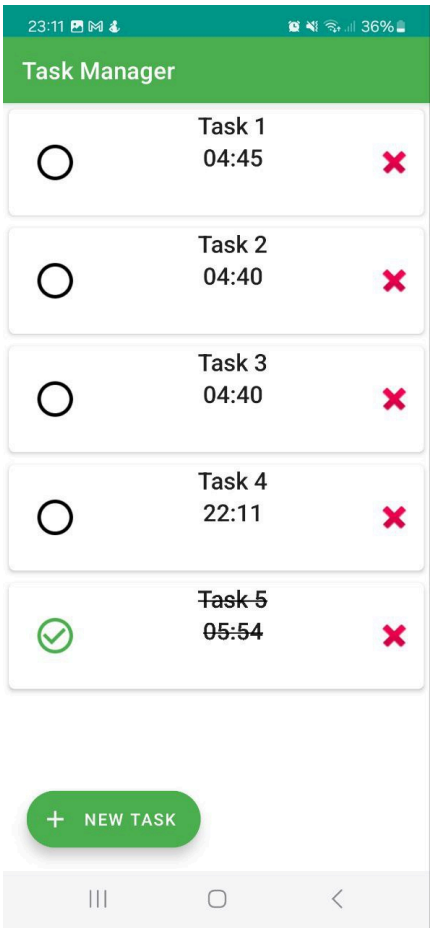
1. https://developer.android.com/

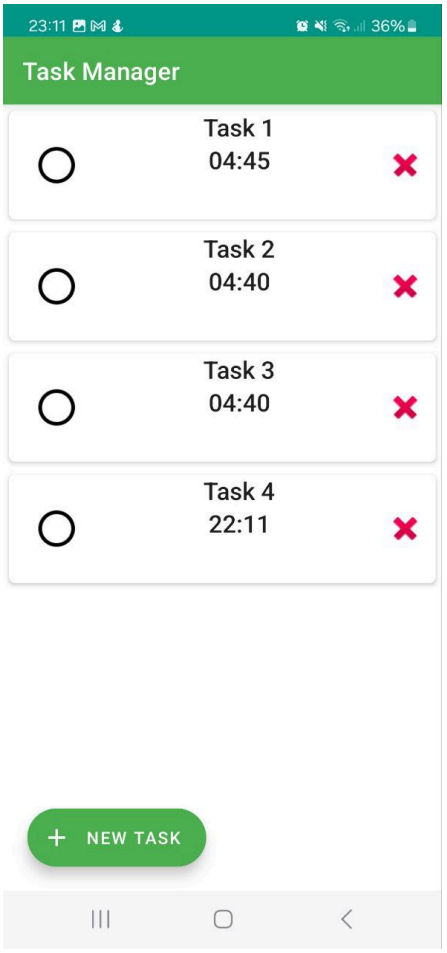
Link to my GitHub repository:

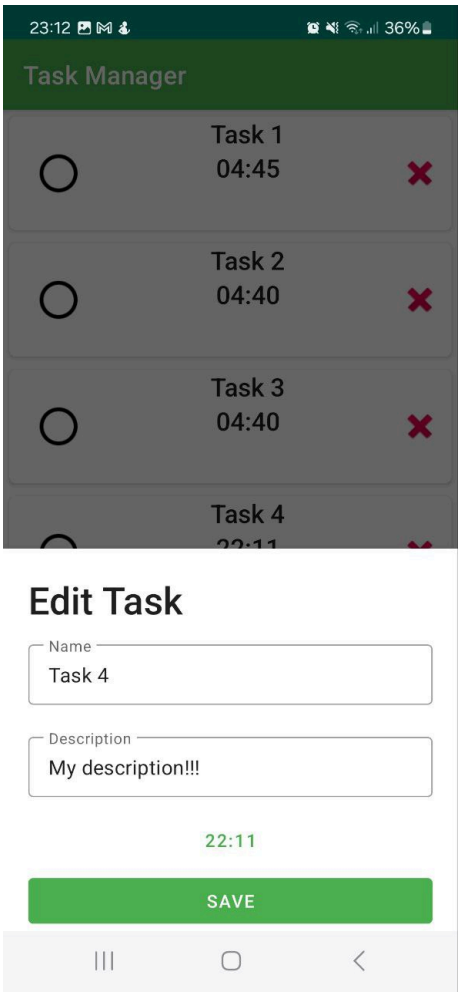
- https://github.com/bozzbala/midterm-task-manager

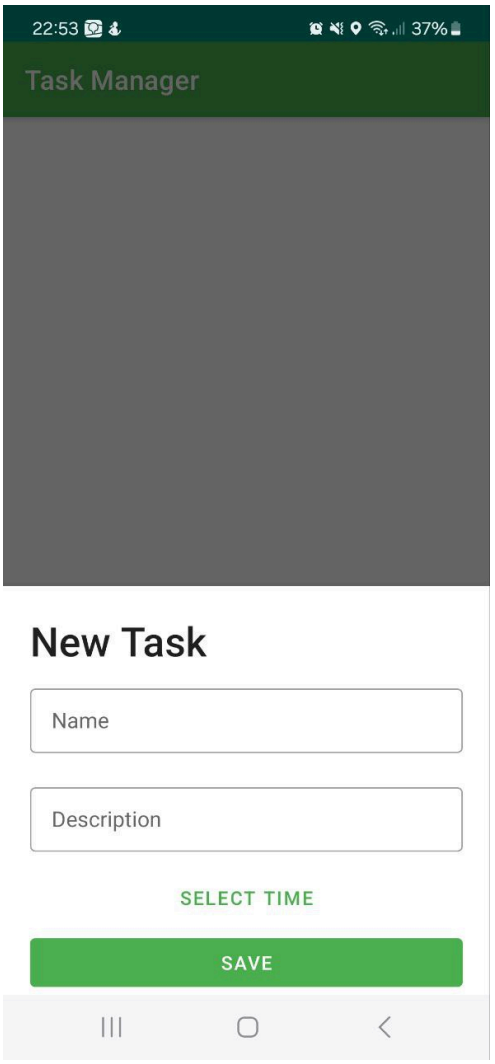
Screenshots




23:11
36%
Task Manager
Task 1
04:45
Task 2
04:40
Task 3
04:40
Task 4
22:11
~~Task 5~~
~~05:54~~
NEW TASK


23:11
36%
Task Manager
Task 1
04:45
Task 2
04:40
Task 3
04:40
Task 4
22:11
NEW TASK


23:12
36%
Task Manager
Task 1
04:45
Task 2
04:40
Task 3
04:40
Task 4
Edit Task
Name
Task 4
Description
My description!!!
22:11
SAVE


22:53
37%
Task Manager
New Task
Name
Description
SELECT TIME
SAVE