REAL ESTATE TODO APP

Table of Contents

Requirements **2**
- Chosen Users/Focus 2
- Functional Requirements: 2

UI Design: **3**
- Design Sketches (Wireframes) 3

Implementation **4**
- 1. NavigationUtil 4
- 2. PickImagesForTodo 5
- 3. Real Estate Database 5
- 4. Orientation Flexible Code 6
- 5. Requesting Location Access from User-: 6
- 6. Geofencing Notification 7
- 7. Reminder Alert Base on Due Date 8
- 8. Reusability 8
- 9. Data Layer Interaction 9

App Brochure **10**
- Adding a New Property 10
- Home Screen after adding properties 12
- Notifications in the Apps 12
- Edge cases that are handled in the app but was not shown in app demo 13

Reflection **14**
- Learning 14
  - 1. Image Capturing 14
  - 2. Location Capture 15
- Future Improvements 15
  - 1. Improving Task Efficiency: Using the Travelling Salesman Problem to Sort Property Tasks 15
  - 2. Synchronized Completion of Property Related Tasks and the Main Property Task: 15
  - 3. Deletion of Sub-Task: 16

Work allocation **16**
- Atharva Sunil Kulkarni 16
- Faiq Iqbal 16
- Sagar Honnenahalli Somashekhar 17
- Manu Kenchappa Junjanna 17
- Jeet Singhvi Pramod 17
- Vivek Shravan Tate 18

Bibliography **18**

Requirements

Chosen Users/Focus

The primary user of our Kotlin-based mobile application is Real Estate agents who sell properties to clients. These agents handle multiple properties scattered across different locations, each carrying its own distinct set of tasks at various stages. When these property-related tasks are not streamlined, chaos can quickly take over. Solving this problem is our main focus and what we are determined to address with our app.

The central requirement for the application is to provide the Real Estate Agent with an efficient task management system tailored to property-related activities. This includes the ability to define and manage custom sets of tasks for each property, ensuring a systematic approach to property management.

Functional Requirements:

1. A custom set of properties and tasks is involved in selling the property-:

   The application allows the user to create and manage a personalized set of tasks for each property. This customization is essential to accommodate the diverse activities associated with the property selling process.

2. Date and Time Reminder-:

   Users have the ability to set date and time reminders for tasks. The application notifies the user when a scheduled task is approaching, ensuring timely execution of property-related activities.

3. Location-Based Notifications-:

   The application provides location-based notifications, alerting the user with the task list when they are in proximity to a property. This feature enhances efficiency by ensuring that relevant tasks are addressed when the users are physically present at the property location.

4. Image Upload-:

   Users are able to upload images related to properties either from the gallery or by camera capture. This facilitates visual documentation and aids in property assessment and management.

5. Task Sorting-:

   The application offers the functionality to sort properties based on date, priority, and location. This ensures a flexible and organized view of the tasks, allowing the user to prioritize and manage their workload effectively.

6. Update Tasks-:

The application also provides the ability to update the defined tasks as needed. The edit functionality is present for the main and sub tasks.

7. Prioritizing the Task-:

   The user can assign priority to tasks according to the urgency in completion of tasks.

UI Design:

The initial design sketches of the application are given below, along with the functionality and features. This is an essential step in the design process, providing a visual blueprint that guides the development of the application and ensuring a more efficient and collaborative design and development process. These sketches can represent early-stage ideas, user interface concepts, and design iterations.

Design Sketches (Wireframes)

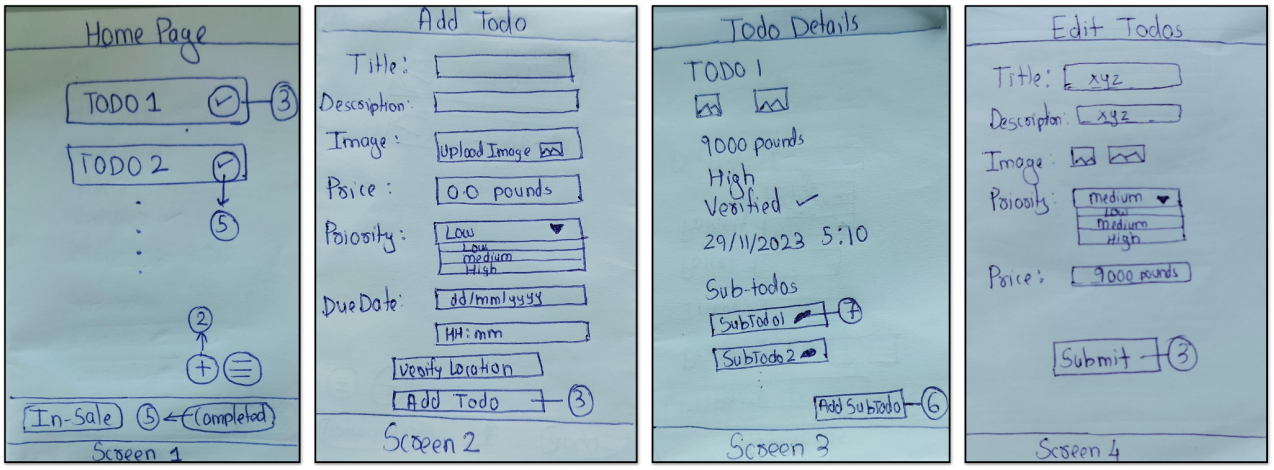


Fig 1.1 : UI WireFrame for Home Page, Add Todo, Todo Details, Edit Todo

Note: In our user focus context, 'Todo' represents a ‘Property’, while 'Sub Todo' stands for ‘Tasks within that property’. This refers to future aspects.

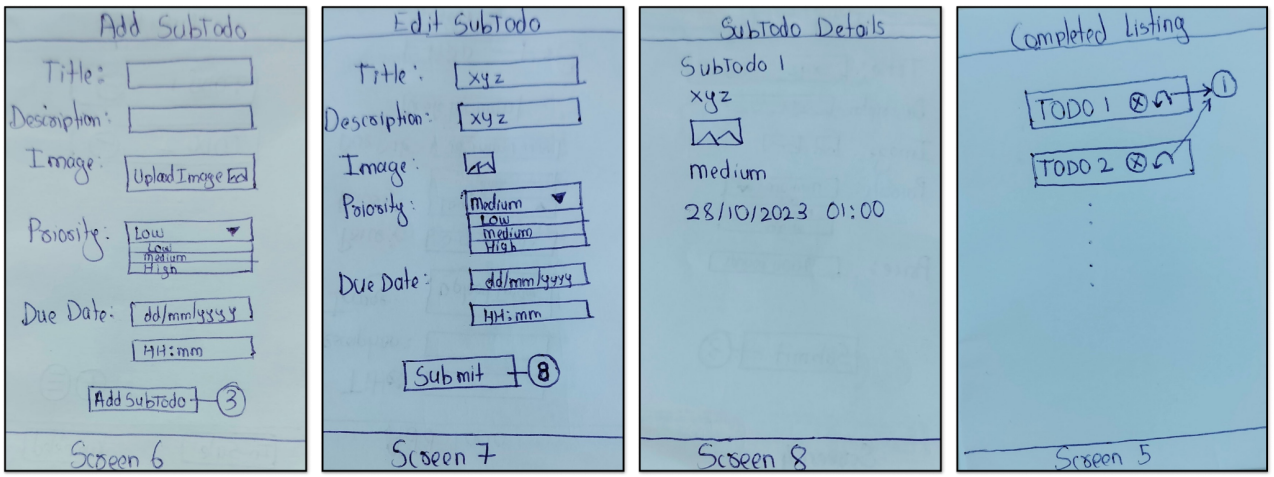


Fig 1.2 : UI WireFrame for Add SubTodo, Edit SubTodo, Sub Todo Details, Completed Listing

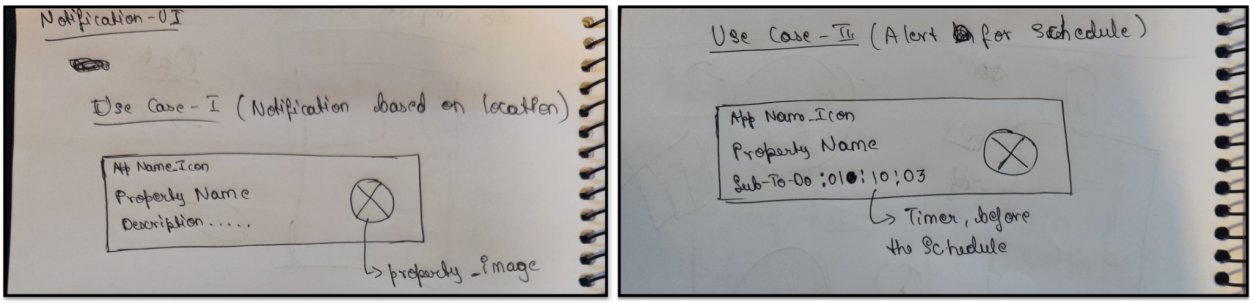


Fig 1.3 : UI WireFrame for Location based Notification and Reminder Notification for scheduled task

Implementation

1. NavigationUtil

NavigationUtil is built using the Singleton Pattern. This helps the app navigate using a NavHostController [3] instance. It also uses an enum class called Screen, which ensures precise navigation by referring to screen names. This way, it cuts down on potential naming errors when making navigation calls .

```
mkenchappajunjanna1 +1
22  enum class Screen {
23      MainScreen, AddTodos, AddOrEditSubToDo, DetailsScreen, SubTodoDetails, EditTodo,EditSubTodo, PreDefinedSubTask
24  }
25
    mkenchappajunjanna1 +1
26  object NavigationUtil {
27      lateinit var navController: NavHostController
28
29      // initializing the nav controller before using it
        mkenchappajunjanna1
30      fun init(navController: NavHostController) {
31          this.navController = navController;
32      }
33
34      // navigate to the given Enum
        mkenchappajunjanna1
35      fun navigateTo(screen: Screen) {
36          navController.navigate(screen.name)
37      }
38
        mkenchappajunjanna1
39      fun navigateTo(screen: String) {
40          navController.navigate(screen)
41      }
42
        mkenchappajunjanna1 +1
43      fun goBack() {
44          navController.popBackStack();
45      }
46  }
```

Fig 2.1 NavigationUtil java/com/team2/todo/utils/NavigationUtil.kt

2. PickImagesForTodo

This feature allows the user to upload images from his external storage in the Add Todos Screen. This class also makes use of bitmaps for storing images. Here, when the user chooses to upload images for the first time from external storage, he/she will be asked to allow the app to access the images through media permissions and then the image upload takes place. rememberLauncherForActivityResult lifecycle method is used for launching the image picker.

```
fun PickImagesForTodo(bitmapList: (List<Bitmap>) -> Unit) {
    val context = LocalContext.current
    var imageUris by remember {
        mutableStateOf<List<Uri>>(emptyList())
    }
    var bitmaps by remember {
        //Bitmap is similar to a pixel of an image
        mutableStateOf<List<Bitmap>>(emptyList())
    }
    val launcher = rememberLauncherForActivityResult(
        contract =
        ActivityResultContracts.GetMultipleContents()
    ) { uris: List<Uri>? ->
        uris?.let { it: List<Uri>
            imageUris = it
            bitmaps =
                it.map { uri ->
                    uri.path?.let { it1 -> Log.d( tag: "Image", it1) }

                    if (Build.VERSION.SDK_INT < 28) {
                        MediaStore.Images.Media.getBitmap(context.contentResolver, uri) ^map
                    } else {
                        val source = ImageDecoder.createSource(context.contentResolver, uri)
                        ImageDecoder.decodeBitmap(source) ^map
                    }
                }
        }
    }
```

Fig 2.2 java/com/team2/todo/screens/add_todo/ui_components/PickImagesForTodo

3. Real Estate Database

The below database class (Fig 2.3) serves as the main access point for the underlying connection to our app's persisted data. It incorporates entities such as Todo, Subtodo and Images while employing a singleton pattern to ensure a single database instance. It also utilizes type converters for complex data types. Thread safety, allowing offline access to cache data in structured tables have been addressed [1].

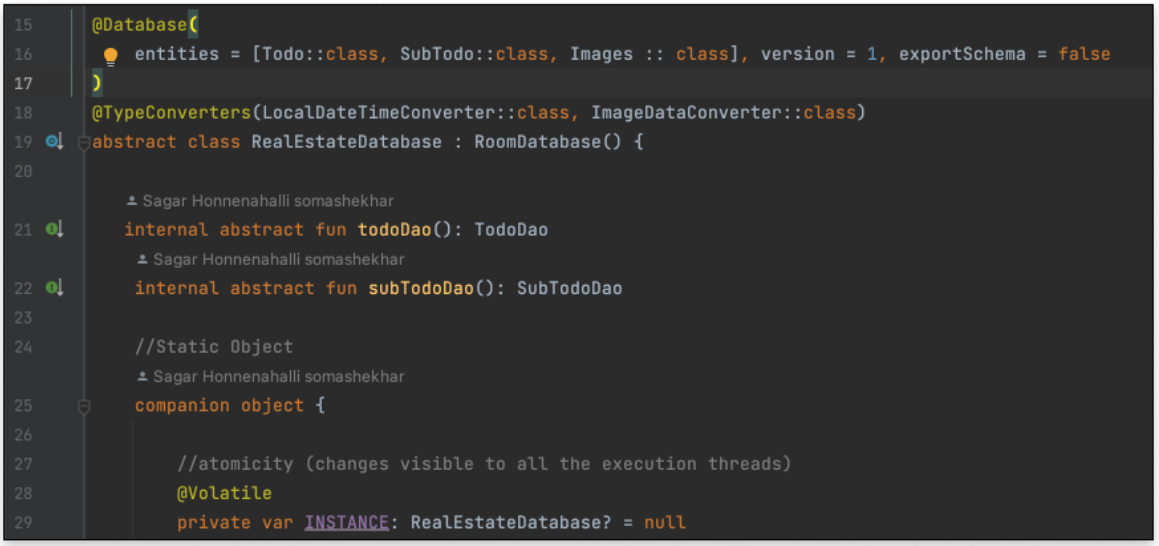


Fig 2.3 java/com/team2/todo/data/RealEstateDatabase.kt

4. Orientation Flexible Code

The rememberSaveable function is used for the all fields in AddProperty/SubTasks and EditProperty/SubTasks screen which is useful for saving the values entered by the user even in case of orientation change of the Android Device

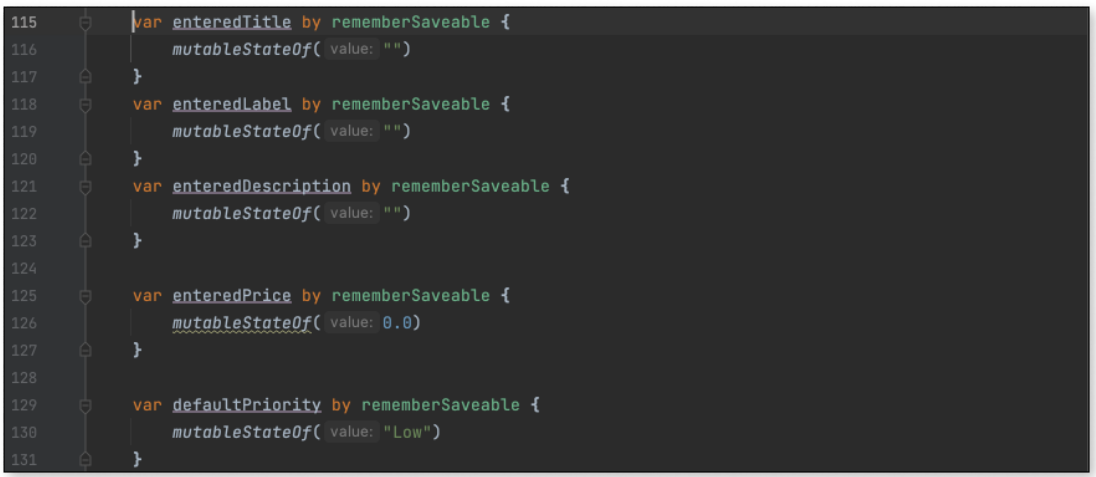
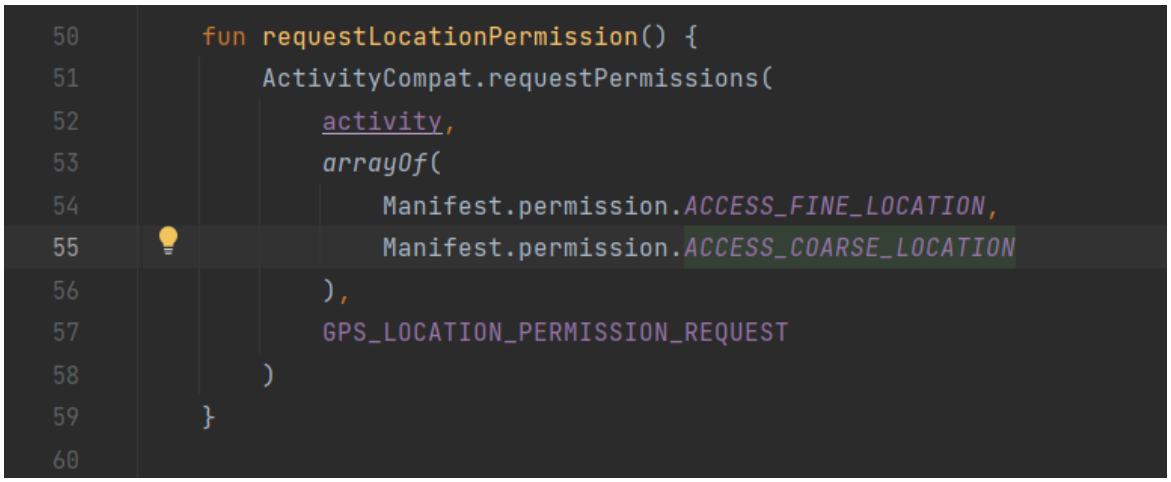


Fig 2.4 AddTodos java/com/team2/todo/screens/add_todo/AddTodos.kt

5. Requesting Location Access from User-:

This functionality is used for requesting location permissions from the user when he installs and opens the app for the first time.It either asks the user to provide precise location or approximate location.



```
50      fun requestLocationPermission() {
51          ActivityCompat.requestPermissions(
52              activity,
53              arrayOf(
54                  Manifest.permission.ACCESS_FINE_LOCATION,
55                  Manifest.permission.ACCESS_COARSE_LOCATION
56              ),
57              GPS_LOCATION_PERMISSION_REQUEST
58          )
59      }
60
```

Fig 2.5 LocationUtil java/com/team2/todo/utils/LocationUtil.kt

6. Geofencing Notification

The Code Snippet in fig 2.6 demonstrates how we trigger notifications using geofencing. This logic specifically applies to properties marked as 'InSaleProperty,' indicating they are currently active. The process involves calculating the distance between the user's current location and the locations of these properties.[5] If the distance falls within the defined THRESHOLD_DSITANCE (set at 500 meters in this instance), a notification is automatically triggered for the respective property. This approach ensures that users receive timely alerts when they are in proximity to these active properties.

```
110                 if (!isNotificationShown) {
111                     inSalePropertyList.value.forEach { it ->
112                         run { this: CoroutineScope
113
114                             if (((it.todo.latitude
115                                     ?: 0.0) != 0.0 || (it.todo.longitude
116                                     ?: 0.0) != 0.0)
117                             ) {
118                                 var distance = GeoFenceUtil.calculateDistance(
119                                     latitude: it.todo.latitude ?: 0.0,
120                                     longitude: it.todo.longitude ?: 0.0,
121                                     LocationUtil.currentLocation!!
122                                 )
123                                 if (distance <= THRESHOLD_DISTANCE) {
124
125                                     NotificationUtil.showGeoFencingNotification(
126                                         property = it
127                                     );
128
129                                     isNotificationShown = true
130                                 }
131                             }
132                         }
133                     }
```

Fig 2.6 PropertyListViewModel
java/com/team2/todo/screens/listing/view_model/PropertyListViewModel.kt

7. Reminder Alert Base on Due Date

Within the code snippet below, we first verify if the due date aligns with today's date. If it does, we then calculate the time difference between the current date and the property's due date. We check if this difference is less than a day (equivalent to 24 * 60 minutes). In cases where multiple properties fall within this time threshold, we prioritize the property with the closest due date. This approach aims to guide users by highlighting the property with the most imminent deadline, signaling that it should be attended to first.

```
74          inSalePropertyList.value.forEachIndexed { index, it ->
75              val dueDateTodo = it.todo.dueDate
76              // if the due date is not today then skip
77              if (!AppUtil.isToday(dueDateTodo)) {
78                  return@forEachIndexed;
79              }
80              // if today then calculate the difference in minute to find the closest task for current time
81              if (dueDateTodo != null) {
82                  val currentDateTime = LocalDateTime.now()
83
84                  val currentDueDateDifference =
85                      ChronoUnit.MINUTES.between(currentDateTime, dueDateTodo)
86
87                  if (currentDueDateDifference >= 0 && currentDueDateDifference <= MINUTES_IN_DAY) {
88                      count++
89                      if (currentDueDateDifference < previousClosestDateDifference) {
90                          previousClosestDateDifference = currentDueDateDifference
91                          closestDuePropertyIndex = index
92                      }
93                  }
94              } else {
95                  println("Fetched Due date is empty for property at index $index!!")
96              }
97          }
```

Fig 2.7 PropertyListViewModel
java/com/team2/todo/screens/listing/view_model/PropertyListViewModel.kt

8. Reusability

The function shown in Fig 2.8 illustrates the use of reusability which we have applied for developing 4 screens- Add Property, Add SubTask, Edit Property and Edit SubTask. There are 3 boolean variables passed as function arguments. When isEdit is passed with value true, the Add Todos Screen gets reused as Edit Todos Screen. Similarly when we pass isSubTodo variable as true, the Screen acts as Add SubTodo Screen with the fields dynamically changed related to SubTodos. When the isEditubTodo is set as true, the Screen works for Edit SubTodo screen allowing the user to update values of fields related to SubTodos.

```
81  fun AddTodos(
82      isSubTodo: Boolean = false,
83      todoid: Long = 0,
84      //isEdit when set to true will allow editing of todos
85      isEdit: Boolean = false,
86      //isEditSubTodo when set to true will allow editing of subtodos
87      isEditSubTodo: Boolean = false
88  ) {
```

Fig 2.8 AddTodos java/com/team2/todo/screens/add_todo/AddTodos.kt

9. Data Layer Interaction

The below Fig 2.9 depicts the interaction between the classes associated with the room database and the ViewModel.

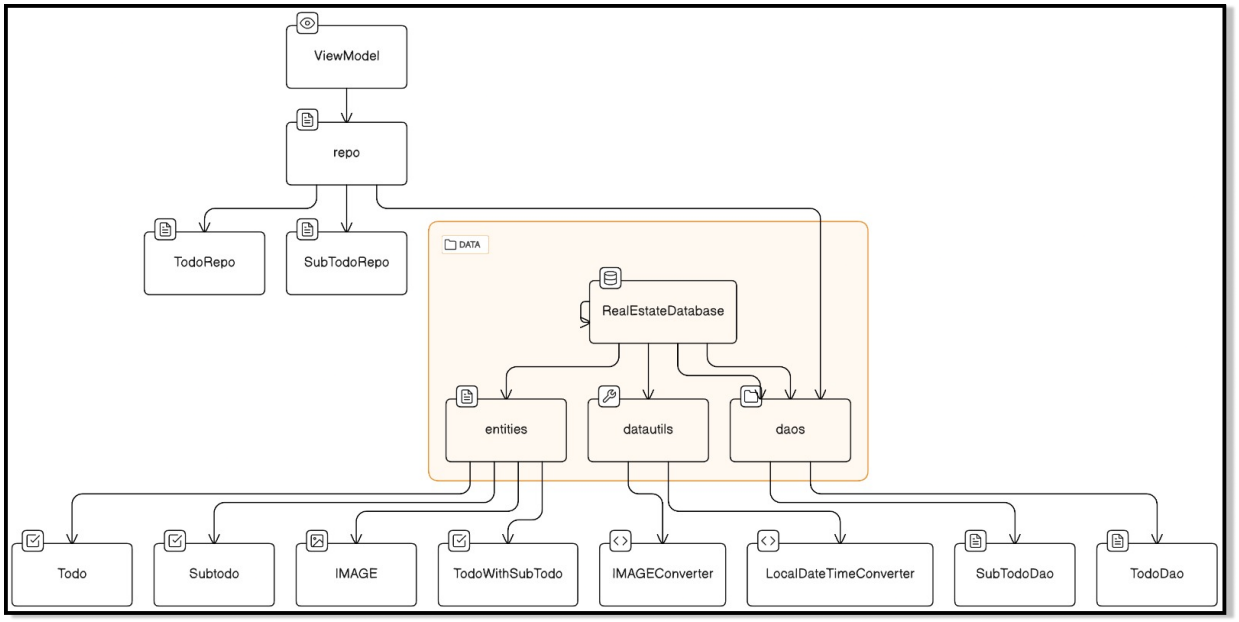


Fig 2.9 Data Layer Interaction

App Brochure

Adding a New Property

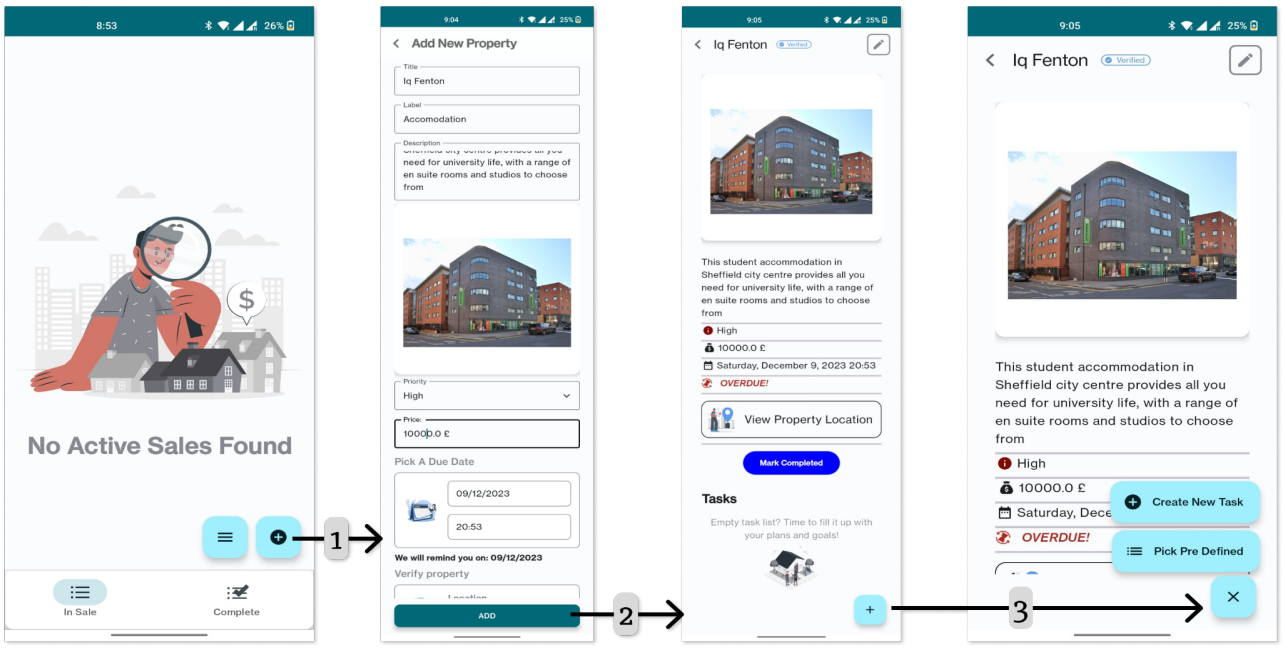


Fig 3.1- Flowchart of Adding a new Property

The application opens to a user-friendly interface featuring a comprehensive list of properties, each accompanied by relevant icons within the property label. For new users, an initial screen communicates, ‘No Active Sales found’, and a prominently displayed ‘+’ icon enables the addition of new properties. The incorporation of essential details such as title, label, description, priority, price, due date, and location enhances the user's ability to organize and manage their properties effectively.

Upon creating a new property, the added property gets seamlessly integrated into the list, intelligently sorted based on the date of creation by default. Clicking on a specific property in the list reveals a detailed view, where users can both review and modify property particulars. Notably, a ‘Mark Completed’ button is prominently featured within this view, facilitating the transition of a property or task to a specialized section known as the ‘Completed List’.

The 'Mark Completed' functionality streamlines the process of acknowledging task fulfillment, subsequently relocating the property to the ‘Completed List’. This dedicated section serves as a repository for all completed tasks, offering users a historical perspective on their accomplishments. The app's intuitive design fosters a seamless and efficient experience, catering to users' needs in property and task management.

Adding tasks for property

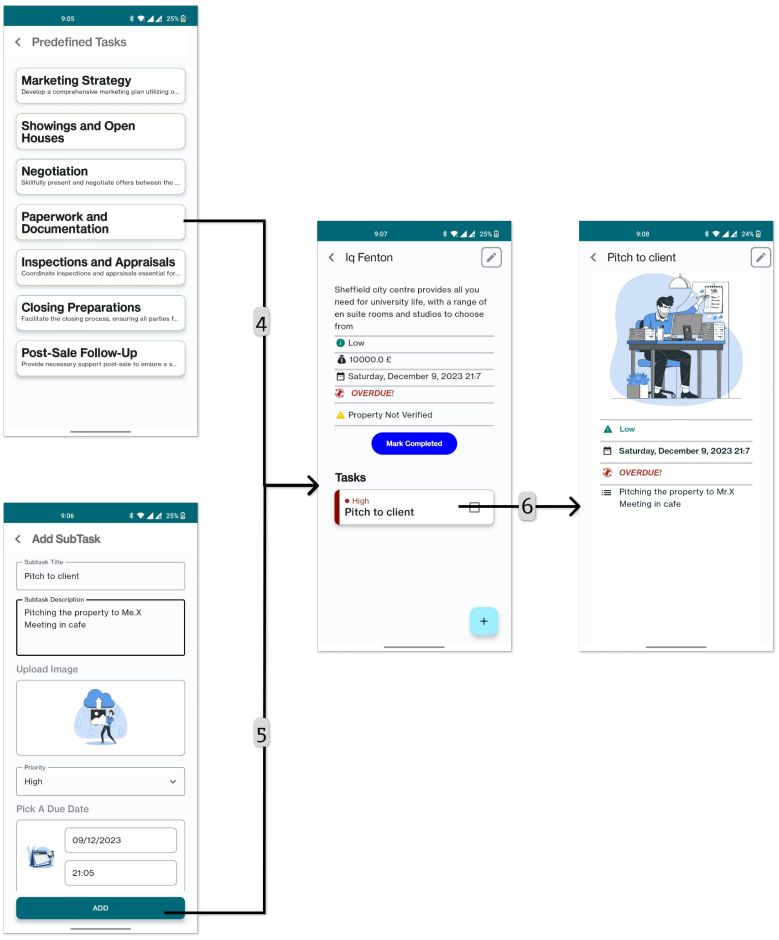


Fig 3.2- Flowchart of Adding SubTasks for a Property

In addition to the property management features, the application incorporates a dynamic task creation functionality accessed through a user-friendly '+' icon. Clicking this icon opens a task creation interface, providing users with two distinct options: selecting from a predefined set of tasks or crafting a custom task based on individual requirements.

Tasks, once added, seamlessly integrate into the property's column, displaying relevant details such as image, priority, and due date. This integration enhances the user's ability to organize and monitor specific tasks within the context of each property.

Moreover, a thoughtful feature includes the display of an 'Overdue' tag when a task surpasses its due date. This visual cue serves as a proactive reminder, offering users a comprehensive overview of pending tasks and their respective urgency levels. This strategic implementation empowers users to prioritize and address tasks promptly, ensuring a streamlined and efficient workflow within the application.

Home Screen after adding properties

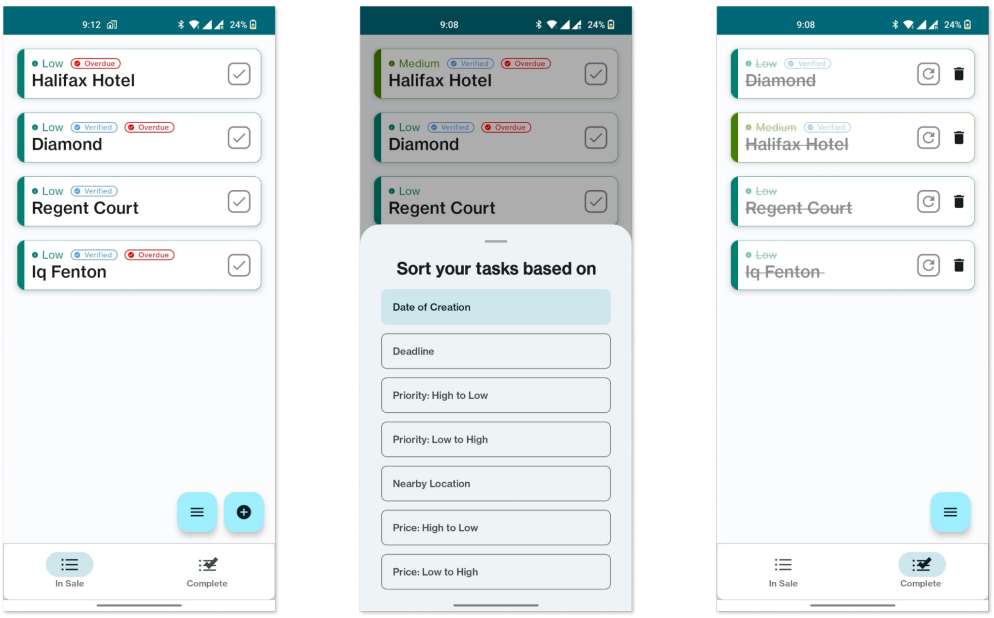


Fig 3.3- Flowchart of Home Screen with added Properties along with filters

The home screen shows the listing of the active tasks referred to as In Sale taks and the list of completed tasks referred to as Complete. Once the user adds a new property as explained above the instance of that property is created in the database and it's shown in the 'In Sale list'. All the respective properties are shown individually as a unique list item. For example, in the above image it can be seen that a property named Halifax Hotel is shown, which was added by the user. The list item displays that it has a low priority and that it has been overdue. Another very important feature of the app that can be used from the home screen is the sorting of the properties based on certain parameters such as date of creation, due date, priority and price. By default the sorting is done based on the date of creation.

Notifications in the Apps

Notifications are a significant part of the application. Geolocation based notifications are generated regarding the active tasks whenever the user is in the vicinity of 500m in the radius of that location to remind the user to complete the tasks associated with that property e.g. in the picture above when the user approaches the Fenton property an alert is generated stating the user has 4 pending tasks related to this property [4]. When the user clicks the button 'Inspect & Complete Task', the user is redirected to the details screen of that particular property showcasing all tasks associated with that property. From there the user can then check the details of those tasks and manage them accordingly.

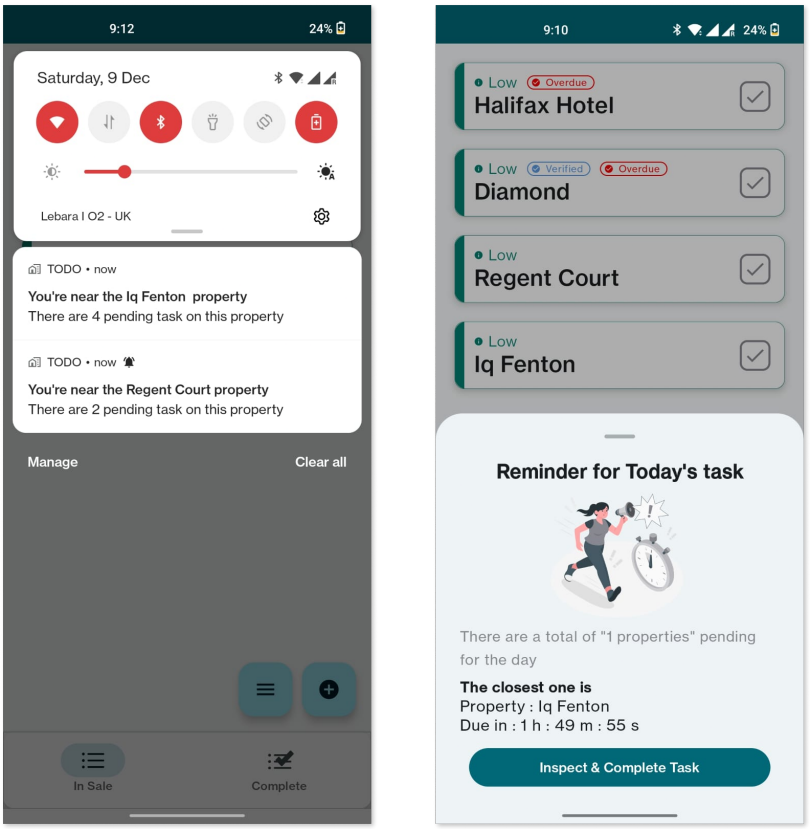
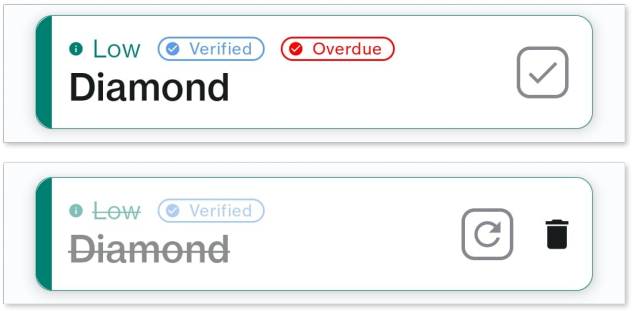


Fig 3.4- Reminder and Pending Tasks Notifications

Edge cases that are handled in the app but was not shown in app demo

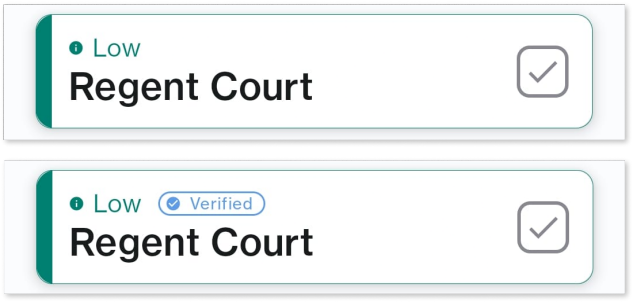
1. **Task Overdue**

If a user fails to meet the deadline for task completion, an 'Overdue' tag will be displayed on the task card in the In-Sale listing page. Once the task is marked as completed and transferred to the Completed listing page, all information is preserved, except the removal of the overdue tag. This tag is no longer necessary upon task completion.



2. **Verified Location Tag**

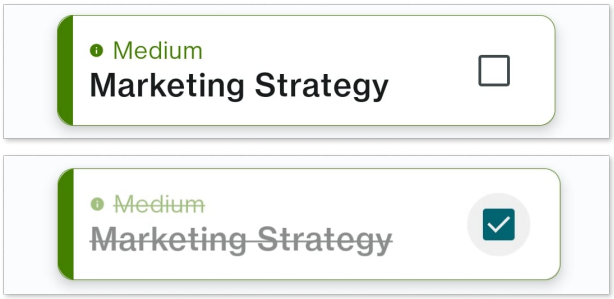
When a user adds a property, they must verify its authenticity by physically visiting the location. The 'Verify Property' button is available on



the property addition page. Once the property is successfully verified, a ‘Verified Property’ tag is shown on the property card in both listing pages. If the property is not yet verified, users can still add it, but the verified tag will not be displayed on the card. Users have the option to visit the property later for verification.

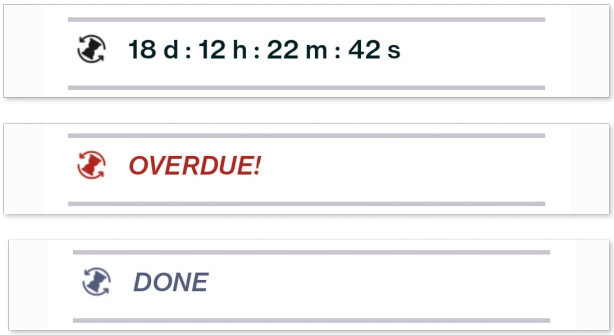
3. **Completed Task Styling**

When all tasks associated with a property are completed, the property is transferred from the ‘In Sales’ property list to the ‘Completed’ list, and the style of the property card is altered. This change includes applying a strikethrough to all the text present on the card.



4. **Due Date and Status Condition**

If a property task is overdue, an additional label appears on the property task details page, providing the user with information about the current status of the property.



Reflection

Learning

1. **Image Capturing**

In our app, users can upload property or subtask-related images using their device's camera [2] or local storage. To enable this feature, we delved into Android's ‘rememberLauncherForActivityResult’ method. This method creates a launcher to fetch selected images from the user's device. We also explored ‘ActivityResult Contracts’ to request images and used ‘RequestMultiplePermissions()’ from the ‘ActivityResultContracts’ class to check user permissions.

Initially, we stored the chosen images, both from the gallery and captured by the camera, as strings of ContentURIs in our Room database. However, a challenge arose when these gallery picked images were not persisting after app restarts. To solve this, we dove into research and experimented with storing image data as byte

arrays. This adjustment ensured the images remained persistent throughout our app's lifecycle.

The pivotal realization was that ContentURIs act as temporary pointers to data managed by a ContentProvider. When the app terminates, the associated data is released by the ContentProvider, making the ContentURI invalid. By storing images as byte arrays, we overcame this challenge of maintaining image persistence seamlessly.

2. **Location Capture**

   During the implementation of Location Capture functionality, we encountered a challenging situation. The onRequestPermissionsResult method was deprecated in previous Android versions, necessitating a delay between permission grant and the initiation of location updates. To address this issue, we implemented a solution: if permissions are already granted, location updates commence immediately. If not, we initiate them in the onResume method of the MainActivity. This approach effectively resolves the timing disparity between permission grant and the start of location updates.

Future Improvements

The following points have been identified for consideration in enhancing the application experience:

1. **Improving Task Efficiency: Using the Travelling Salesman Problem to Sort Property Tasks**

   We wanted gearing up to enhance productivity by introducing a smart feature.It revolves around pinpointing where you are, drawing a circle around it, and identifying nearby properties as key stops for your tasks. Imagine it like planning the best route for a delivery person, except it's for your property tasks! Once we've gathered these tasks, we will organize them based on their importance and when they need to be done. Then, we'll present it all on a map. This map will be your guiding star, helping you figure out which task to tackle first, saving you time and effort as you move between different properties.

2. **Synchronized Completion of Property Related Tasks and the Main Property Task:**

   In the present state of our application, a notable discrepancy exists wherein independently marking all the property related tasks does not trigger the automatic completion of the main property task as a whole, and conversely, marking the main property task as complete does not synchronize with the task related to it. This observed incongruence is slated for resolution in our upcoming enhancements. The proposed solution aims to optimize user experience and streamline task management functionality within the application.

3. **Deletion of Sub-Task:**

   In light of the current functionality gap within our application, whereby users are limited to deleting only main property tasks and unable to delete individual tasks related to that property, it is imperative that we address this limitation in our future development efforts. Accordingly, we propose to elevate the deletion of tasks associated with the property as a formal functional requirement to enhance the overall usability and flexibility of our application.

Work allocation

Atharva Sunil Kulkarni

1. Created wireframes for Add Property, Edit Property, Add SubTasks, Edit SubTasks Screen along with other team members.
2. Idea proposal of Trello, a team management tool.
3. Add Property Screen Development-:
   a) Initial UI with various components such as DateTimePicker, Dropdown, ImagePicker.
   b) Validation for certain fields such as allowing only 1 word for label property.
   c) Integrating verify location functionality in the screen.
4. Edit Property Screen Development-:
   a) Reuse of AddProperty Screen with Edit Flag set to true.
   b) Displaying values associated with each field and allowing user to edit them.
5. Add SubTask Screen Development-:
   a) Reuse of AddProperty Screen with isSubtask Flag set to true.
   b) Filtering out only a subset of fields related to SubTask from the fields of add property screen.
6. Edit SubTask Screen Development-:
   a) Reuse of AddProperty Screen with isEditSubtask Flag set to true.
   b) Displaying values associated with each field of subtask and allowing user to edit them.

Faiq Iqbal

I was assigned to develop the details screen of the main task. Following is the further breakdown of how I planned and managed the implementation:

1. First of all a simple screen covering all the fields that were to be included in the details screen like description, priority, price etc was created and dummy hardcoded data was added to it.
2. Integration of database to the screen to display real-time data instead of dummy data.
3. A task completion button named Mark as Complete is implemented which marks the task completed and sends it to the completed tasks list.

4. Creation of a lazy list of all the subtasks and displayed them in the details screen. Each list item can be clicked to view the details of that specific subtask.
5. Creation of a dynamic timer counter on the details screen to display the time to due date.
6. Implementation of the location mapping by getting the location coordinates from the database and creating an intent to open google maps and show the exact pin location on google maps.

Sagar Honnenahalli Somashekhar

1. Creation of Database schema.
2. Setting up of the Initial Datalayer (daos, repo, Entities etc,.), implemented data transaction and manipulation logic for the Properties and the Tasks.
3. Implementation of the Type Converters.
4. Added a BottomSheet for the selection of the upload mode (Gallery/Camera) in the Add Property UI.
5. Implementation of Camera capture functionality, persistent storage of the images.
6. Implementation of the Image loader component.
7. Implementation of the countdown timer component wrt due date.
8. Implementation of the Task details screen.
9. Involved in manual testing of the application and fixing bugs that were detected.

Manu Kenchappa Junjanna

1. Code Review and Branch Management
2. Initial Project Setup
3. Implementation of Navigation Controller
4. Integration of notifications and notification push
5. Creation of a template folder for screen creation
6. Screens/BottomSheet to handle loading condition (for async task)
7. Home Screen development
    a. Bottom Navigation Component setup
    b. UI Revamping of Listing Item and Completed Item
    c. Reminder BottomSheet UI
8. Add Property Screen
    a. UI Revamping for components in the Add Property section
9. Property Details Screen development
    a. Addition of a Floating Action Button for Task Creation
10. Implemented Predefined Tasks Screen for selecting repetitive tasks common across different properties
11. Implementation of common composables like CommonAppBar, EmptyList, VerifyByLocationCompose etc

Jeet Singhvi Pramod

1. Idea proposal for user and focus - Real estate broker, managing different properties
2. Worked on the listing screen UI for ‘In-sale’ and ‘Completed’ List

3. Worked on handling the lazy list of tasks and sub-tasks
4. Worked on the logic for handling of tasks which are overdue
5. Involved in bug fixing that were detected as the development progressed
6. Designed the logo of the app

Vivek Shravan Tate

1. Implementation of Location Services
    a. Introduced Location Util
    b. Integrated check and request for Missing Permissions
    c. Resume and Pause location service when app in background
2. Implementation of Filter Functionality
    a. Addition of Floating Action Button for Filter Selection
    b. Addition of Bottom Sheet for Filter List
    c. Introduction and Implementation of 7 Filter Categories
    d. Reloading the Listing Screens post filter selection
3. Implementation of Geo-Fencing
    a. Addition of GeoFence Mechanism
    b. Tune it for location based sorting in filter functionality.
    c. Customised it for location based task alerts.

Bibliography

1. Google.(n.d). Save data in a local database using Room [Online]. Android Developers. [Viewed 30th October 2023]. Available from: https://developer.android.com/training/data-storage/room
2. Google.(n.d). Take photos [Online]. Android Developers. [Viewed 10th November 2023]. Available from: https://developer.android.com/training/camera-deprecated/photobasics
3. Google.(n.d). Navigating with Compose [Online]. Android Developers. Available from: https://developer.android.com/jetpack/compose/navigation
4. Google.(n.d). Create a notification [Online]. Android Developers. Available from: https://developer.android.com/develop/ui/views/notifications/build-notification
5. Stackoverflow. Calculating distance between to location parameters for geofencing [Online]. David George. Available from: https://stackoverflow.com/a/16794680