- **Introduction of your project. Gives an overview of the project, rationale behind the creation of this mobile app, and the requirements.**

Overview of the project: SearchLost App

The SearchLost App is a ground-breaking mobile application created with a humanitarian goal in mind: to aid in the rapid and efficient search for missing or lost individuals. It serves as a technology beacon of hope, bringing communities, volunteers, and resources together to address the critical and sensitive issue of missing persons. The app aims to revolutionize the way missing person situations are handled by offering a platform for cooperation, communication, and coordination among users, authorities, and volunteers through a user-centric approach and innovative capabilities.

Rationale behind the creation of this mobile app

The SearchLost App was developed in response to a compelling need for a modern solution to the issues connected with missing person instances. When it comes to finding missing people, time is of the importance, and traditional approaches, such as physical posters or fliers, might be limited in their reach and efficacy. To address this, the app intends to give an immediate and extensive way of disseminating important information about missing people to the community and necessary authorities, thereby dramatically boosting the likelihood of their safe return. In addition, the app is intended to strengthen communities during times of crisis. Community participation and support are critical in search efforts, and the app does this by allowing users to participate actively in the search process. The app encourages a sense of responsibility and collaborative action by allowing users to make missing person reports and share information, connecting the community in working towards a common goal. Real-time communication and coordination are critical for any effective search activity. The app acts as a primary center for users, volunteers, and authorities to communicate updates and information. This real-time communication accelerates the search process by allowing users to share leads, sightings, and progress, creating an effective and efficient joint effort. The app also includes powerful search features, with sophisticated filtering options that allow users to easily discover pertinent missing person reports. The program reduces the need for manual sorting and enhances targeted search efforts by allowing searches based on certain phrases, regions, or timeframes, increasing the odds of identifying missing individuals quickly.

When dealing with sensitive missing person cases, ensuring privacy and data protection is critical. The app strictly adheres to privacy policies and guidelines, preserving user information and ensuring that contributions are only accessible to authorized search participants. The application makes use of cutting-edge image recognition technology to speed up the identifying process. This revolutionary feature assists users, volunteers, and authorities in verifying sightings by identifying missing people based on images, considerably expediting search operations and increasing the likelihood of a successful resolution. Furthermore, by utilizing the pervasiveness of smartphones and social media, the app provides a broad-reaching awareness campaign. Encouraging users to post missing person reports across numerous platforms expands the app's reach beyond its user base,

improving the likelihood of the missing person being found quickly. Finally, the SearchLost App unites compassion, technology, and community support by offering a modern and efficient platform for locating missing people. The app aims to make a significant and lasting impact in reuniting missing people with their families and loved ones by addressing critical aspects such as timely information sharing, community engagement, real-time communication, advanced search capabilities, data security, image recognition, and extensive awareness.

Requirements: SearchLost App

The SearchLost App aspires to provide a comprehensive and user-friendly tool for dealing with missing person instances. It allows users to submit missing person reports that include vital information such as the individual's name, photo, description, and the date and time of their missing. The application has a real-time communication system that allows users, volunteers, and authorities to exchange updates, leads, and sightings to help with effective search activities. Users may rapidly discover relevant missing person reports due to advanced search and filtering capabilities, and picture recognition technology aids in verifying sightings, speeding up the identification process. The app actively promotes community engagement through social sharing, enabling users to publish missing person reports across several platforms to enhance awareness and the possibility of quickly locating missing people. Individuals of diverse backgrounds can actively participate in the search process via a user-friendly interface intended for ease of navigation. Users can examine specific information on missing person cases and amend and update reports as needed to provide accurate and up-to-date information.

Offline support means that the app's basic features are available even when there is no internet connection, and data synchronization occurs when connectivity is restored. Users can safely upload and save missing person images, and a date and time picker function allow users to specify the missing person's disappearance date and time precisely. The application has a "Share Task List" feature that allows users to share a comprehensive list of all reported missing persons across various communication channels, facilitating teamwork. A thorough backup and data recovery process is designed to protect data integrity, ensuring the app stays resilient in the face of unanticipated occurrences. Comprehensive error handling and recording methods are in place to discover, track, and rectify any issues or defects as soon as possible, ensuring that the program runs smoothly and efficiently. By achieving these goals, the SearchLost App aspires to be a significant tool in the search for missing people, stimulate community engagement, and eventually bring comfort and hope to those touched by such occurrences.

- **Implementation details of the mobile app and justification of the approaches.**

MainActivity.kt:

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

    toolbarMainActivity.title = "Searching For Lost Person"
    toolbarMainActivity.subtitle = "The snow to send carbon"
    setSupportActionBar(toolbarMainActivity)

    recyclerView.setHasFixedSize(true)
    recyclerView.layoutManager = LinearLayoutManager( context: this@MainActivity)

    db = DatabaseHelper( context: this@MainActivity)

    getAllTasks()

    fab.setOnClickListener { it: View!
        startActivity(Intent( packageContext: this@MainActivity, TaskRegistrationActivity::class.java))
    }
}
```

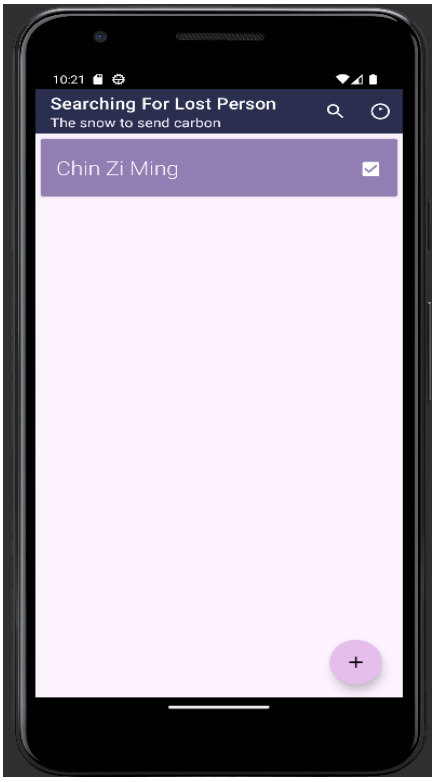

Figure As

As the figure As above shown, onCreate: This function is called when the MainActivity is created. It sets up the layout, initializes the toolbar, and configures the RecyclerView. It also initializes the DatabaseHelper and retrieves all tasks from the database using TaskDAO. Then, it populates the RecyclerView with the TaskAdapter, displaying the list of tasks.

```
override fun onCreateOptionsMenu(menu: Menu): Boolean {
    menuInflater.inflate(R.menu.search_menu, menu)

    val item = menu.findItem(R.id.action_search)
    val searchView = item.actionView as SearchView
    searchView.setOnQueryTextListener(this)

    val shareItem = menu.findItem(R.id.action_share)
    shareItem.setOnMenuItemClickListener { it: MenuItem
        shareTaskList()
        true ^setOnMenuItemClickListener
    }

    return super.onCreateOptionsMenu(menu)
}
```

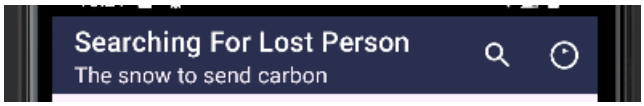

Figure Bs

As the figure Bs above shown, onCreateOptionsMenu: This function is called to inflate the options menu for the MainActivity. It adds the search view and share item to the toolbar. It also sets listeners for search view text changes and the share item click.

```
override fun onBackPressed() {
    val intent = Intent(Intent.ACTION_MAIN)
    intent.addCategory(Intent.CATEGORY_HOME)
    intent.flags = Intent.FLAG_ACTIVITY_NEW_TASK
    startActivity(intent)
}
```

Figure C

As the figure C above shown, onBackPressed: This function is overridden to customize the behavior when the back button is pressed. It moves the app to the background instead of exiting it.

Additional features:

```
override fun onQueryTextSubmit(query: String): Boolean {
    Log.e( tag: "Submitted search query", query)
    performSearch(query)
    return true
}

override fun onQueryTextChange(newText: String): Boolean {
    Log.e( tag: "Search query change", newText)
    performSearch(newText)
    return true
}
```

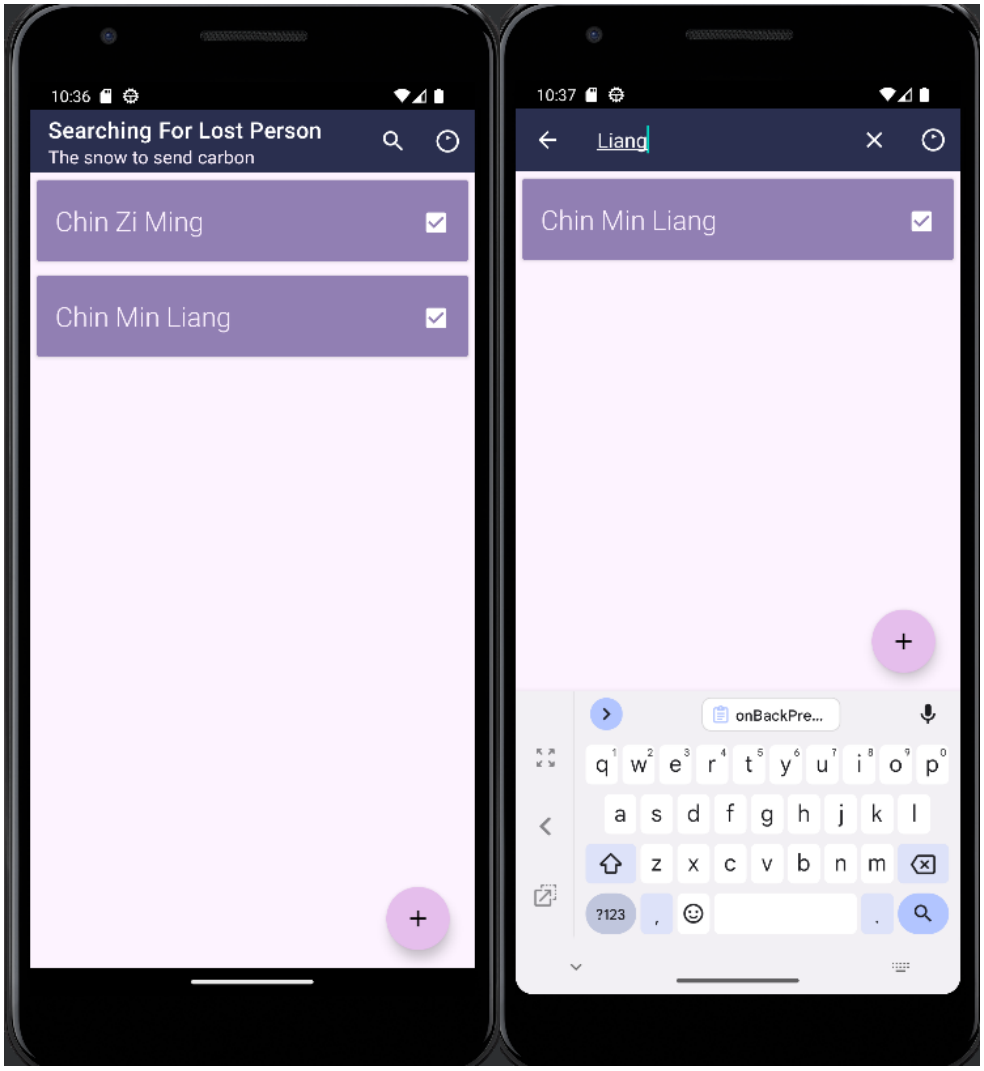

Figure Ds

As the figure Ds shown, onQueryTextSubmit and onQueryTextChange: These functions are callbacks for search view text changes. When the user submits or changes the search

query, they call the performSearch function to filter the tasks based on the search keyword and update the RecyclerView.

```
private fun getAllTasks() {
    taskList = TaskDAO().getAllTasks(db)
    adapter = TaskAdapter( mContext: this@MainActivity, taskList, db)
    recyclerView.adapter = adapter
}

private fun performSearch(searchKeyword: String) {
    taskList = TaskDAO().searchTasks(db, searchKeyword)
    adapter = TaskAdapter( mContext: this@MainActivity, taskList, db)
    recyclerView.adapter = adapter
}
```

Figure E

As Figure E shown, getAllTasks and performSearch: These functions interact with the database using TaskDAO to retrieve and filter tasks, respectively. They update the taskList and RecyclerView with the new data.

Additional features:

```
private fun getAllTasksAsString(): String {
    val taskList = TaskDAO().getAllTasks(db)
    val stringBuilder = StringBuilder()

    for (task in taskList) {
        stringBuilder.append("Person Image: ${task.taskPhotoUrl}\n")
        stringBuilder.append("Person Name: ${task.taskTitle}\n")
        stringBuilder.append("Missing Person Details: ${task.taskName}\n")
        stringBuilder.append("Missing Date and Time: ${task.taskDateTime}\n\n")
    }

    return stringBuilder.toString()
}

private fun shareTaskList() {
    val tasks = getAllTasksAsString()
    val sendIntent = Intent().apply { this: Intent
        action = Intent.ACTION_SEND
        type = "text/plain"
        putExtra(Intent.EXTRA_SUBJECT, value: "List of Tasks")
        putExtra(Intent.EXTRA_TEXT, tasks)
    }
    val shareIntent = Intent.createChooser(sendIntent, title: "Share Tasks List")
    startActivity(shareIntent)
}
```

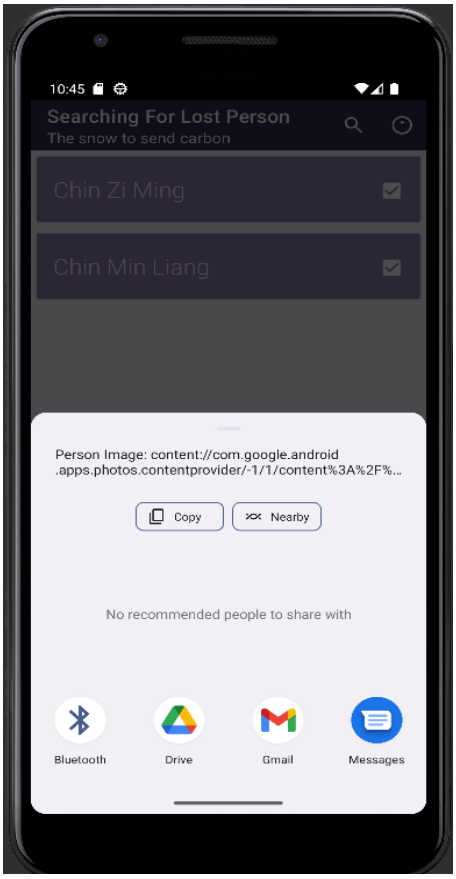

Figure Fs

As the Figure Fs shown, getAllTasksAsString and shareTaskList: These functions handle sharing the list of tasks as plain text. getAllTasksAsString retrieves all tasks from the database and formats them as a string. shareTaskList creates an intent to share the task list using the built-in Android sharing functionality.

Task.kt:

```
import java.io.Serializable

data class Task(
    var taskId: Int,
    var taskTitle: String,
    var taskName: String,
    var taskPhotoUrl: String?,
    var taskDateTime: String?

) : Serializable
```

Figure G

As the Figure G shown, this file defines the data class Task representing a single missing person report. It contains properties like taskId, taskTitle, taskName, taskPhotoUrl, and taskDateTime.

TaskAdapter.kt:

```
inner class CardViewHolder(view: View) : RecyclerView.ViewHolder(view) {
    val cardView: CardView = view.findViewById(R.id.cardView)
    val textView: TextView = view.findViewById(R.id.textViewTaskName)
    val imageView: ImageView = view.findViewById(R.id.imageViewDelete)
}
```

```
override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): CardViewHolder {
    val view = LayoutInflater.from(mContext).inflate(R.layout.task_row_layout, parent, attachToRoot: false)
    return CardViewHolder(view)
}

override fun getItemCount(): Int {
    return taskList.size
}
```

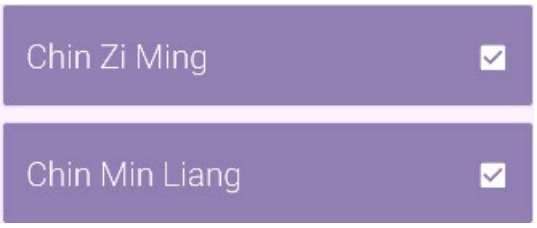

Figure Hs

As the Figure Hs shown, CardViewHolder: This inner class defines the view holder for each item in the RecyclerView. It holds references to the cardView, textView (task title), and imageView (delete icon). onCreateViewHolder: This function creates a CardViewHolder by inflating the task_row_layout.xml. getItemCount: This function returns the number of items in the taskList.

Additional features:

```
override fun onBindViewHolder(holder: CardViewHolder, position: Int) {
    val task = taskList[position]

    holder.textView.text = task.taskTitle

    holder.imageView.setOnClickListener { it: View!
        Toast.makeText(
            mContext,
            text: "${task.taskTitle} has been found. Thank you for giving a helping hand",
            Toast.LENGTH_SHORT
        ).show()

        TaskDAO().deleteTask(db, task.taskId)

        taskList = TaskDAO().getAllTasks(db)
        notifyDataSetChanged()
    }

    holder.cardView.setOnClickListener { it: View!
        val intent = Intent(mContext, TaskDetailActivity::class.java)
        intent.putExtra( name: "task", task)
        mContext.startActivity(intent)
    }
}
```

Figure Is

In the Figure Is shown, onBindViewHolder: This function binds data to the CardViewHolder for a particular position in the taskList. It sets the task title and click listeners for the delete icon and task card.

TaskDAO.kt:

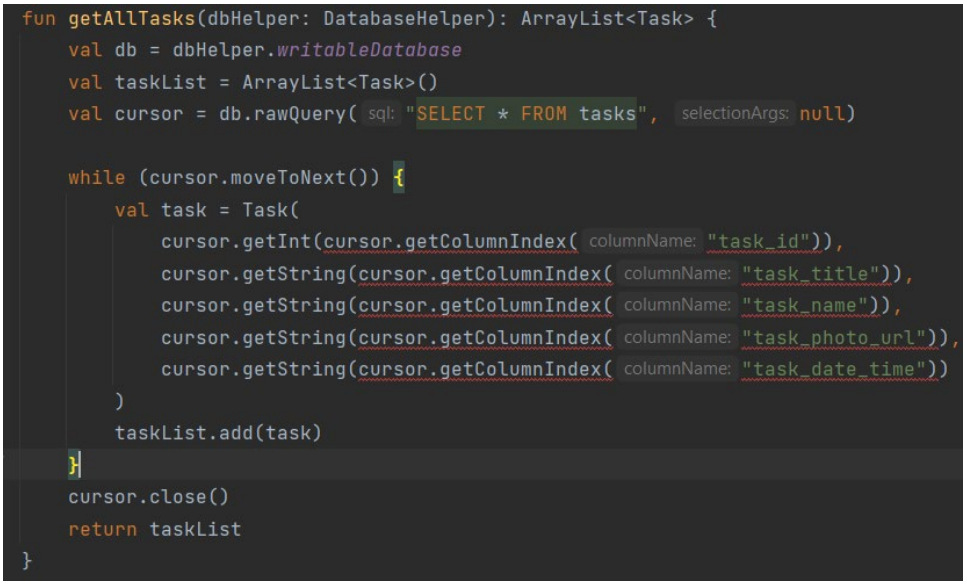

Figure J

As the Figure J shown, getAllTasks: This function retrieves all tasks from the database and returns them as an ArrayList of Task objects.

```
fun searchTasks(dbHelper: DatabaseHelper, searchKeyword: String): ArrayList<Task> {
    val db = dbHelper.writableDatabase
    val taskList = ArrayList<Task>()
    val cursor = db.rawQuery(
        sql: "SELECT * FROM tasks WHERE task_title LIKE '%$searchKeyword%'",
        selectionArgs: null
    )

    while (cursor.moveToNext()) {
        val task = Task(
            cursor.getInt(cursor.getColumnIndex( columnName: "task_id")),
            cursor.getString(cursor.getColumnIndex( columnName: "task_title")),
            cursor.getString(cursor.getColumnIndex( columnName: "task_name")),
            cursor.getString(cursor.getColumnIndex( columnName: "task_photo_url")),
            cursor.getString(cursor.getColumnIndex( columnName: "task_date_time"))
        )
        taskList.add(task)
    }
    cursor.close()
    return taskList
}
```

Figure K

In Figure K shown, searchTasks: This function searches for tasks with titles containing the given searchKeyword and returns the matching tasks as an ArrayList of Task objects.

```
fun deleteTask(dbHelper: DatabaseHelper, taskId: Int) {
    val db = dbHelper.writableDatabase
    db.delete( table: "tasks",  whereClause: "task_id=?", arrayOf(taskId.toString()))
    db.close()
}

fun addTask(dbHelper: DatabaseHelper, taskTitle: String, taskName: String): Int {
    val db = dbHelper.writableDatabase
    val values = ContentValues()
    values.put(DatabaseHelper.COLUMN_TASK_TITLE, taskTitle)
    values.put(DatabaseHelper.COLUMN_TASK_NAME, taskName)

    val insertedId = db.insertOrThrow(DatabaseHelper.TABLE_NAME,  nullColumnHack: null, values)
    db.close()
    return insertedId.toInt()
}
```

Figure L

As the Figure L shown, deleteTask: This function deletes a task with a given taskId from the database. addTask: This function adds a new task to the database with the provided taskTitle and taskName, returning the new task's taskId.

```kotlin
fun getImageURL(dbHelper: DatabaseHelper, taskId: Int): String? {
    val db = dbHelper.readableDatabase
    val cursor = db.query(
        DatabaseHelper.TABLE_NAME,
        arrayOf(DatabaseHelper.COLUMN_TASK_PHOTO_URL),
        selection: "${DatabaseHelper.COLUMN_TASK_ID}=?",
        arrayOf(taskId.toString()),
        groupBy: null,
        having: null,
        orderBy: null
    )
    var imageURL: String? = null

    if (cursor.moveToFirst()) {
        imageURL = cursor.getString(cursor.getColumnIndex(DatabaseHelper.COLUMN_TASK_PHOTO_URL))
    }

    cursor.close()
    return imageURL
}
```

```kotlin
fun insertImageURL(dbHelper: DatabaseHelper, taskId: Int, imageURL: String) {
    val db = dbHelper.writableDatabase
    val values = ContentValues()
    values.put(DatabaseHelper.COLUMN_TASK_PHOTO_URL, imageURL)

    db.update(
        DatabaseHelper.TABLE_NAME,
        values,
        whereClause: "${DatabaseHelper.COLUMN_TASK_ID}=?",
        arrayOf(taskId.toString())
    )

    db.close()
}
```

Figure Ms

As the Figure Ms shown, getImageURL and insertImageURL: These functions handle getting and updating the taskPhotoUrl for a specific task in the database, respectively.

```kotlin
fun updateDateTime(dbHelper: DatabaseHelper, taskId: Int, dateTime: String) {
    val db = dbHelper.writableDatabase

    val values = ContentValues()
    values.put(DatabaseHelper.COLUMN_TASK_DATE_TIME, dateTime)

    db.update(
        DatabaseHelper.TABLE_NAME,
        values,
        whereClause: "${DatabaseHelper.COLUMN_TASK_ID}=?",
        arrayOf(taskId.toString())
    )
    db.close()
}

fun updateTask(dbHelper: DatabaseHelper, taskId: Int, taskTitle: String, taskName: String, taskDateTime: String) {
    val db = dbHelper.writableDatabase

    val values = ContentValues()
    values.put(DatabaseHelper.COLUMN_TASK_TITLE, taskTitle)
    values.put(DatabaseHelper.COLUMN_TASK_NAME, taskName)
    values.put(DatabaseHelper.COLUMN_TASK_DATE_TIME, taskDateTime)

    db.update(
        DatabaseHelper.TABLE_NAME,
        values,
        whereClause: "${DatabaseHelper.COLUMN_TASK_ID}=?",
        arrayOf(taskId.toString())
    )
    db.close()
}
```

Figure N

As the Figure N shown, updateDateTime and updateTask: These functions handle updating the taskDateTime and other task properties for a specific task in the database, respectively.

TaskRegistrationActivity.kt:

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

    toolbarTaskRegistrationActivity.title = "Save The Missing Person Details!"
    setSupportActionBar(toolbarTaskRegistrationActivity)

    db = DatabaseHelper( context: this@TaskRegistrationActivity)

    buttonSelectDateTime.setOnClickListener { it: View!
        showDateTimePicker()
    }

    button.setOnClickListener { it: View!
        val galleryIntent = Intent(Intent.ACTION_PICK, MediaStore.Images.Media.EXTERNAL_CONTENT_URI)
        startActivityForResult(galleryIntent, REQUEST_IMAGE_PICKER)
    }

    buttonSave.setOnClickListener { it: View!
        val taskTitle = editTextTaskTitle.text.toString()
        val taskName = editTextTaskName.text.toString()

        saveTask(taskTitle, taskName)
    }
}
```

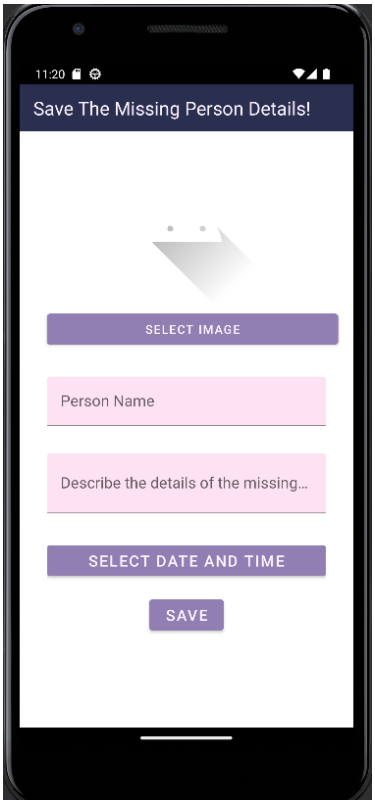

Figure Os

As the Figure Os shown, onCreate: This function is called when the TaskRegistrationActivity is created. It sets up the layout and toolbar and initializes the DatabaseHelper. It also sets click listeners for the date and time picker and image selection buttons.

```
private fun showDateTimePicker() {
    val calendar = Calendar.getInstance()
    val currentYear = calendar.get(Calendar.YEAR)
    val currentMonth = calendar.get(Calendar.MONTH)
    val currentDay = calendar.get(Calendar.DAY_OF_MONTH)
    val currentHour = calendar.get(Calendar.HOUR_OF_DAY)
    val currentMinute = calendar.get(Calendar.MINUTE)

    val datePickerDialog = DatePickerDialog(
        context: this,
        DatePickerDialog.OnDateSetListener { _: DatePicker, year: Int, monthOfYear: Int, dayOfMonth: Int ->
            selectedDate = "$dayOfMonth/${monthOfYear + 1}/$year"
            showTimePicker(currentHour, currentMinute)
        },
        currentYear,
        currentMonth,
        currentDay
    )


    datePickerDialog.show()
}
```

```
private fun showTimePicker(currentHour: Int, currentMinute: Int) {
    val timePickerDialog = TimePickerDialog(
        context: this,
        TimePickerDialog.OnTimeSetListener { _: TimePicker, hourOfDay: Int, minute: Int ->
            selectedTime = String.format("%02d:%02d", hourOfDay, minute)
            buttonSelectDateTime.text = "$selectedDate, $selectedTime"
        },
        currentHour,
        currentMinute,
        is24HourView: true
    )

    timePickerDialog.show()
}
```

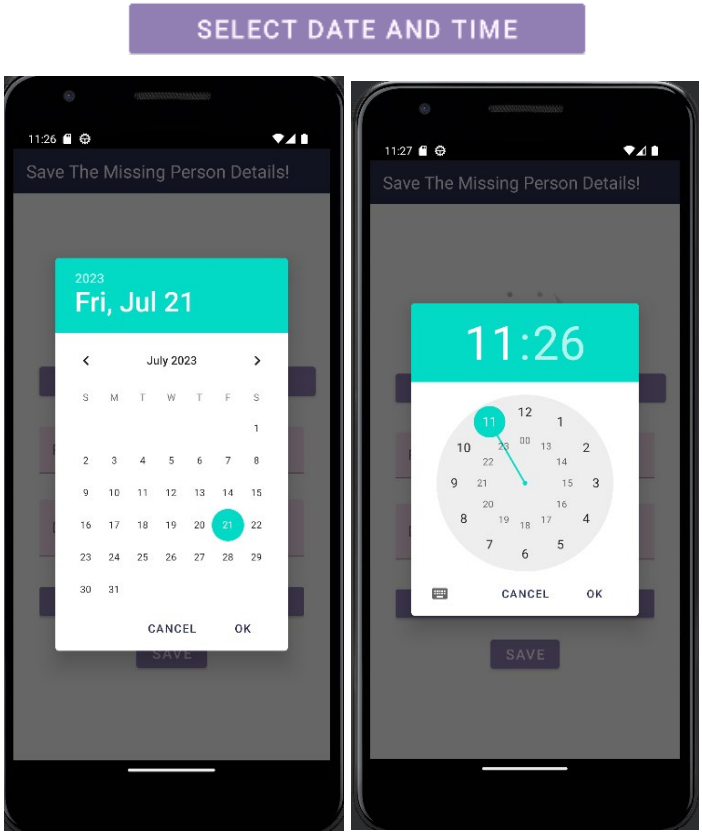

Figure Ps

showDateTimePicker, showTimePicker: These functions handle showing the DatePickerDialog and TimePickerDialog to select the date and time of the missing person's disappearance and update the dateButton with the selected date and time.

```kotlin
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)

    if (requestCode == REQUEST_IMAGE_PICKER && resultCode == Activity.RESULT_OK) {
        val selectedImageUri = data?.data
        photourl = selectedImageUri.toString()

        val imageView: ImageView = findViewById(R.id.imageView)
        photourl?.let { it: String
            Glide.with( activity: this).load(it).placeholder(R.drawable.photoholder).into(imageView)
        } ?: run { this: TaskRegistrationActivity
            imageView.setImageResource(R.drawable.photoholder)
        }
    }
}
```

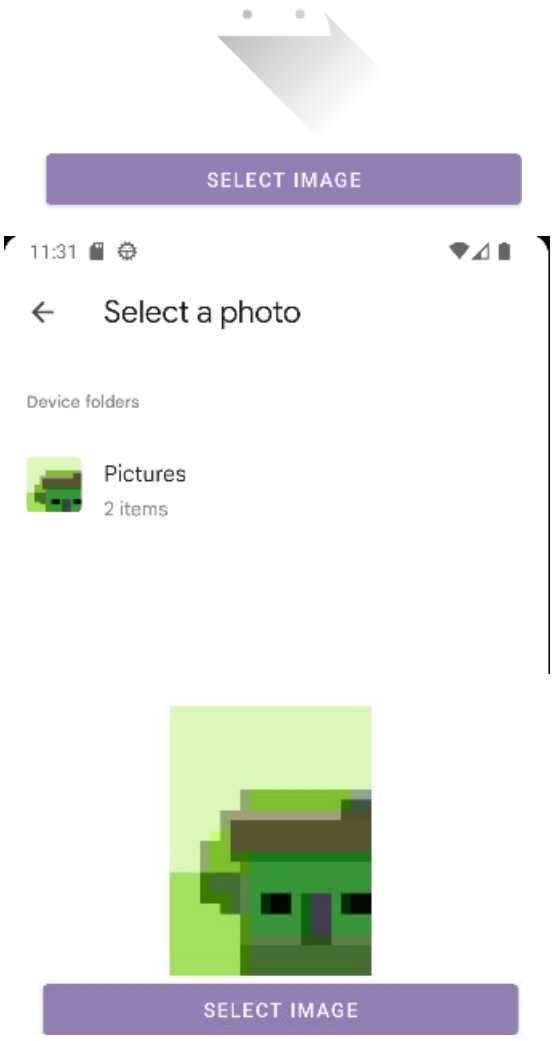

Figure Qs

As the Figure Qs shown, onActivityResult: This function is called when the user selects an image from the gallery. It retrieves the selected image's URI and updates the imageView with the chosen image.

```
private fun saveTask(taskTitle: String, taskName: String) {
    if (taskTitle.isBlank() || taskName.isBlank()) {
        Toast.makeText( context: this,  text: "Please enter both task title and task name", Toast.LENGTH_SHORT).show()
        return
    }

    if (photourl == null) {
        Toast.makeText( context: this,  text: "Please select an image", Toast.LENGTH_SHORT).show()
        return
    }

    if (selectedDate == null || selectedTime == null) {
        Toast.makeText( context: this,  text: "Please select date and time", Toast.LENGTH_SHORT).show()
        return
    }

    Log.e( tag: "Task Save",  msg: "Title: $taskTitle, Name: $taskName")


    val taskId = TaskDAO().addTask(db, taskTitle, taskName)


    db.insertImageURL(taskId, photourl!!)


    val dateTime = "$selectedDate, $selectedTime"
    TaskDAO().updateDateTime(db, taskId, dateTime)

    startActivity(Intent( packageContext: this@TaskRegistrationActivity, MainActivity::class.java))
}
```

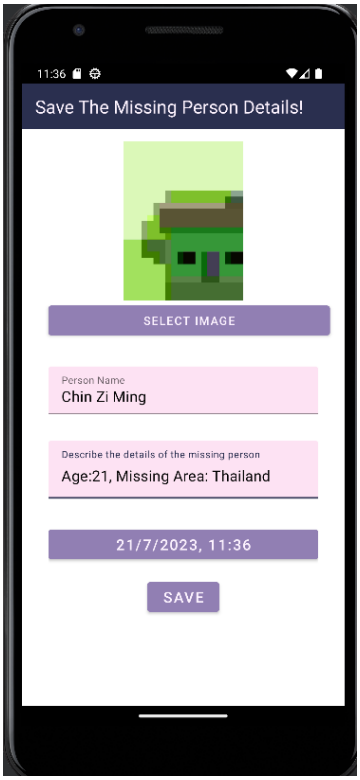

Figure Rs

As the Figure Rs shown, saveTask: This function saves the new task to the database with the provided taskTitle, taskName, taskDateTime, and taskPhotoUrl.

TaskDetailActivity.kt:

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

    toolbarTaskDetailActivity.title = "Update The Missing Person Details!"
    setSupportActionBar(toolbarTaskDetailActivity)

    db = DatabaseHelper( context: this@TaskDetailActivity)

    task = intent.getSerializableExtra( name: "task") as Task

    editTextTaskTitle.setText(task.taskTitle)
    editTextTaskName.setText(task.taskName)

    val imageURL = TaskDAO().getImageURL(db, task.taskId)
    if (!imageURL.isNullOrEmpty()) {
        Glide.with( activity: this).load(imageURL).placeholder(R.drawable.photoholder).into(imageView)
    } else {
        imageView.setImageResource(R.drawable.photoholder)
    }

    dateButton = findViewById(R.id.dateButton)
    calendar = Calendar.getInstance()
    dateButton.text = task.taskDateTime

    dateButton.setOnClickListener { it: View!
        showDateTimePicker()
    }
```

```
    button.setOnClickListener { it: View!
        val galleryIntent = Intent(Intent.ACTION_PICK, MediaStore.Images.Media.EXTERNAL_CONTENT_URI)
        startActivityForResult(galleryIntent, REQUEST_IMAGE_PICKER)
    }

    buttonUpdate.setOnClickListener { it: View!
        val taskTitle = editTextTaskTitle.text.toString()
        val taskName = editTextTaskName.text.toString()
        val taskDateTime = dateButton.text.toString()
        updateTask(task.taskId, taskTitle, taskName, taskDateTime)
    }
}
```

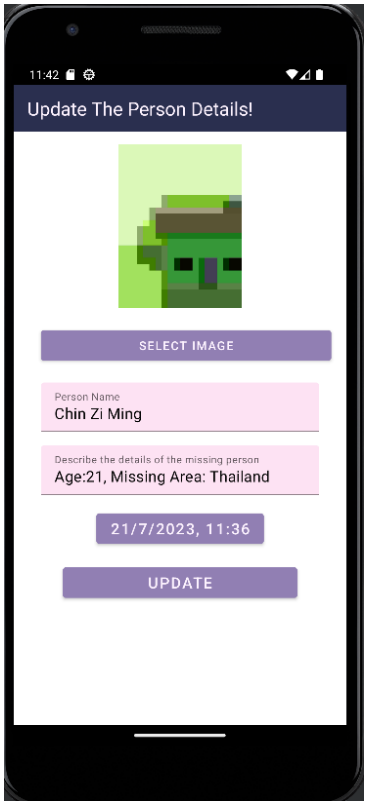

Figure Ss

As Figure Ss shown, onCreate: This function is called when the TaskDetailActivity is created. It sets up the layout and toolbar, initializes the DatabaseHelper, and retrieves the Task object passed through the intent. Then, it populates the UI elements with the task details.

```
private fun showDateTimePicker() {
    val dateListener = DatePickerDialog.OnDateSetListener { _: DatePicker, year: Int, month: Int, dayOfMonth: Int ->
        calendar.set(Calendar.YEAR, year)
        calendar.set(Calendar.MONTH, month)
        calendar.set(Calendar.DAY_OF_MONTH, dayOfMonth)


        showTimePicker()
    }

    val datePickerDialog = DatePickerDialog(
        context: this, dateListener,
        calendar.get(Calendar.YEAR), calendar.get(Calendar.MONTH), calendar.get(Calendar.DAY_OF_MONTH)
    )

    datePickerDialog.show()
}
```

```
private fun showTimePicker() {
    val timeListener = TimePickerDialog.OnTimeSetListener { _: TimePicker, hourOfDay: Int, minute: Int ->
        calendar.set(Calendar.HOUR_OF_DAY, hourOfDay)
        calendar.set(Calendar.MINUTE, minute)

        updateDateButton()
    }

    val timePickerDialog = TimePickerDialog(
        context: this, timeListener,
        calendar.get(Calendar.HOUR_OF_DAY), calendar.get(Calendar.MINUTE),  is24HourView: true
    )

    timePickerDialog.show()
}

private fun updateDateButton() {
    val sdf = SimpleDateFormat( pattern: "yyyy-MM-dd HH:mm", Locale.getDefault())
    val dateString = sdf.format(calendar.time)
    dateButton.text = dateString
}
```

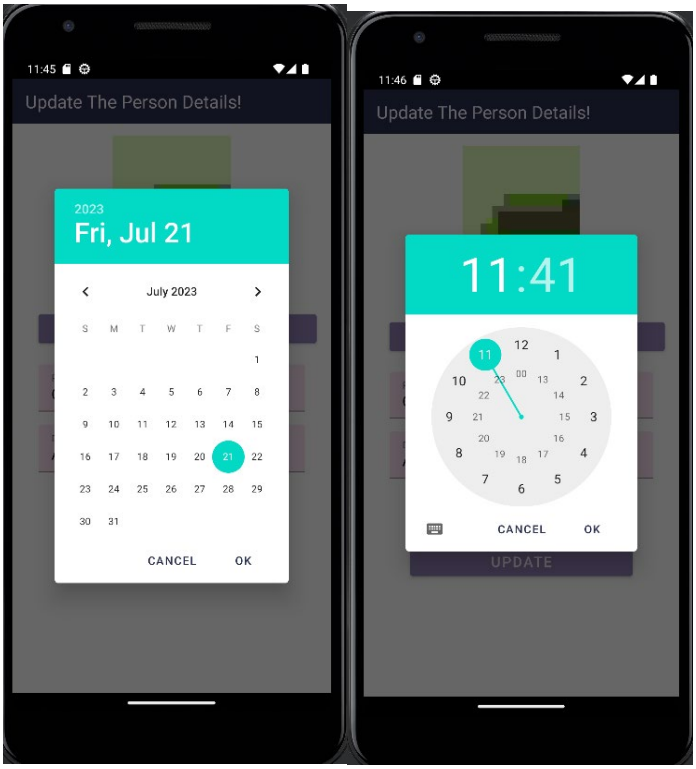

Figure Ts

As the Figure Ts shown, showDateTimePicker, showTimePicker, and updateDateButton: These functions handle showing the DatePickerDialog and TimePickerDialog to select the date and time of the missing person's disappearance. They also update the dateButton with the selected date and time.

```
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)

    if (requestCode == REQUEST_IMAGE_PICKER && resultCode == Activity.RESULT_OK) {
        selectedImageUri = data?.data
        val imageView: ImageView = findViewById(R.id.imageView)
        selectedImageUri?.let { it: Uri
            Glide.with( activity: this).load(it).placeholder(R.drawable.photoholder).into(imageView)
        } ?: run { this: TaskDetailActivity
            imageView.setImageResource(R.drawable.photoholder)
        }
    }
}
```

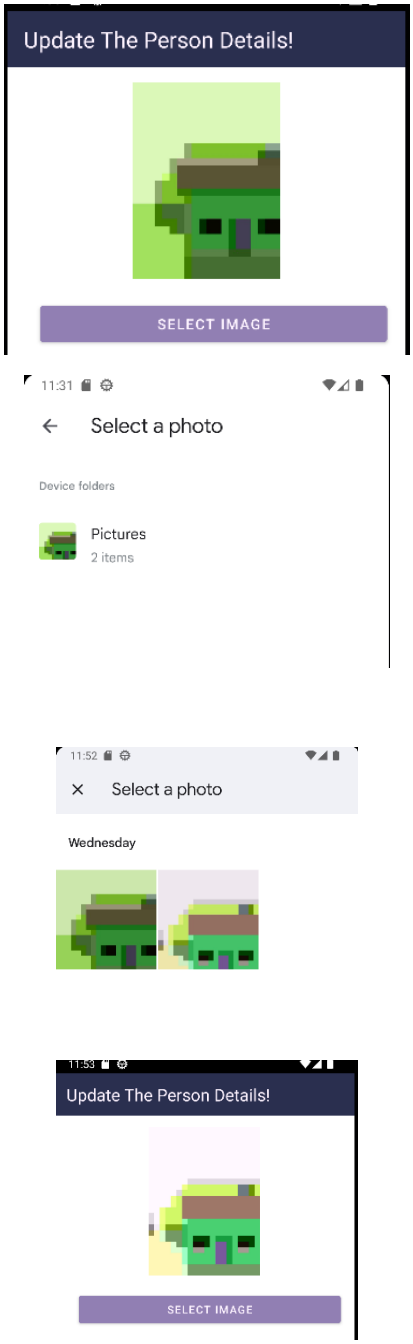

Figure Us

As the Figure Us shown, onActivityResult: This function is called when the user selects an image from the gallery. It retrieves the selected image's URI and updates the imageView with the chosen image.

```
private fun updateTask(taskId: Int, taskTitle: String, taskName: String, taskDateTime: String) {
    if (taskTitle.isBlank() || taskName.isBlank()) {
        Toast.makeText( context: this,  text: "Please enter both task title and task name", Toast.LENGTH_SHORT).show()
        return
    }

    if (task.taskTitle == taskTitle && task.taskName == taskName && task.taskDateTime == taskDateTime) {
        Toast.makeText( context: this,  text: "No changes made to the task", Toast.LENGTH_SHORT).show()
        return
    }

    Log.e( tag: "Update Task",  msg: "$taskId - $taskTitle - $taskName")


    TaskDAO().updateTask(db, taskId, taskTitle, taskName, taskDateTime)


    selectedImageUri?.let { it: Uri
        val imageURL = it.toString()
        TaskDAO().insertImageURL(db, taskId, imageURL)
    }

    startActivity(Intent( packageContext: this@TaskDetailActivity, MainActivity::class.java))
    finish()
}
```

Figure Vs

As the Figure Vs shown, updateTask: This function updates the task details in the database with the new taskTitle, taskName, and taskDateTime. It also updates the taskPhotoUrl if a new image is selected.

DatabaseHelper.kt:

```
class DatabaseHelper(context: Context) : SQLiteOpenHelper(context, DATABASE_NAME, factory: null, DATABASE_VERSION)
```

Figure W

As the Figure W shown, this line defines the DatabaseHelper class, which extends SQLiteOpenHelper. It acts as a helper class to manage the SQLite database for the app. It takes a Context parameter and calls the superclass constructor with the database name (DATABASE_NAME) and version number (DATABASE_VERSION).

```
companion object {
    private const val DATABASE_NAME = "tasks.db"
    private const val DATABASE_VERSION = 5
    const val TABLE_NAME = "tasks"
    const val COLUMN_TASK_ID = "task_id"
    const val COLUMN_TASK_TITLE = "task_title"
    const val COLUMN_TASK_NAME = "task_name"
    const val COLUMN_TASK_PHOTO_URL = "task_photo_url"
    const val COLUMN_TASK_DATE_TIME = "task_date_time"
}
```

Figure X

As the Figure X shown, the companion object block contains constants used in the database operations. It defines the database name, version, table name, and column names for the tasks table. It also includes a column for the task_photo_url and task_date_time.

```
override fun onCreate(db: SQLiteDatabase) {
    val createTableQuery = (
            "CREATE TABLE $TABLE_NAME (" +
                    "$COLUMN_TASK_ID INTEGER PRIMARY KEY AUTOINCREMENT, " +
                    "$COLUMN_TASK_TITLE TEXT NOT NULL, " +
                    "$COLUMN_TASK_NAME TEXT NOT NULL, " +
                    "$COLUMN_TASK_PHOTO_URL TEXT, " +
                    "$COLUMN_TASK_DATE_TIME TEXT" +
                    ")"
            )
    db.execSQL(createTableQuery)
}
```

Figure Y

As the Figure Y shown, the onCreate function is called when the database is created for the first time. It creates the tasks table with the specified columns using the CREATE TABLE SQL query.

```
override fun onUpgrade(db: SQLiteDatabase, oldVersion: Int, newVersion: Int) {
    when {
        oldVersion < 2 -> {
            // If the old version is 1, perform the necessary upgrade to version 2
            val alterTableQuery = (
                    "ALTER TABLE $TABLE_NAME " +
                            "ADD COLUMN $COLUMN_TASK_TITLE TEXT NOT NULL DEFAULT ''"
                    )
            db.execSQL(alterTableQuery)
        }
        oldVersion < 3 -> {
            // If the old version is 1 or 2, perform the necessary upgrade to version 3
            // Add any other database changes for version 3 here
            // For example:
            // db.execSQL("ALTER TABLE $TABLE_NAME ADD COLUMN new_column INTEGER DEFAULT 0")
        }
        oldVersion < 4 -> {
            // If the old version is 1, 2 or 3, perform the necessary upgrade to version 4
            // Add the new column for task photo URL
            val alterTableQuery = (
                    "ALTER TABLE $TABLE_NAME " +
                            "ADD COLUMN $COLUMN_TASK_PHOTO_URL TEXT"
                    )
            db.execSQL(alterTableQuery)
        }
        oldVersion < 5 -> {
            val alterTableQuery = (
                    "ALTER TABLE $TABLE_NAME " +
                            "ADD COLUMN $COLUMN_TASK_DATE_TIME TEXT"
                    )
            db.execSQL(alterTableQuery)
```

Figure Z

The onUpgrade function is called when the database needs to be upgraded, which occurs when the DATABASE_VERSION is incremented. It handles different upgrade scenarios based on the old and new versions of the database.

```
fun insertImageURL(taskId: Int, imageURL: String) {
    val db = this.writableDatabase
    val values = ContentValues()
    values.put(COLUMN_TASK_PHOTO_URL, imageURL)
    db.update(TABLE_NAME, values, whereClause: "$COLUMN_TASK_ID=?", arrayOf(taskId.toString()))
    db.close()
}
```

Figure A1

As the Figure A1 shown, the insertImageURL function inserts the imageURL for a specific task with the given taskId. It updates the task_photo_url column in the database for that task using the update method.

- **Test cases and test results**

Test Case ID	TC-A-001
Objective	Test the add function at the MainActivity.kt
Steps to test	1. Click on the plus button 2. Select Image 3. Input Person Name textview 4. Input Describe the details of the missing person textview 5. Select Date 6. Select Time 7. Click Save button
Data inputs	1. Select image of the missing person 2. Person Name: Chin Zi Ming 3. Describe the details of the missing person: Age:21, Missing Area: Thailand 4. Select Date: 21/7/2023 5. Select Time: 20:54 (pm)
Expected Results	The details of a missing person were added.
Actual Results	Meet the expected results
Test Results	Pass

Test Case ID	TC-A-002
Objective	Test the search function in MainActivity.kt
Steps to test	1. Add few Missing Person Details 2. Search one of the Person Name
Data inputs	Person 1: 1. Select image of the missing person 2. Person Name: Chin Zi Ming 3. Describe the details of the missing person: Age:21, Missing Area: Thailand 4. Select Date: 21/7/2023 5. Select Time: 20:54 (pm) Person 2: 1. Select image of the missing person 2. Person Name: Chin Min Liang 3. Describe the details of the missing person: Age:22, Missing Area: Myanmar 4. Select Date: 11/7/2023 5. Select Time: 10:54 (am) Person 3: 1. Select image of the missing person 2. Person Name: Chin Min Lin

	3. Describe the details of the missing person: Age:25, Missing Area: Malaysia 4. Select Date: 31/7/2023 5. Select Time: 21:54 (pm)
Expected Results	The name is sorted out and found
Actual Results	Meet the expected results
Test Results	Pass

Test Case ID	TC-A-003
Objective	Test the share missing list functions in MainActivity.kt
Steps to test	1. Share the list
Data inputs	Nothing to be inputed
Expected Results	The details generated and ready to be share via other platforms
Actual Results	Meet the expected results
Test Results	Pass

Test Case ID	TC-A-004
Objective	Test the items in MainActivity.kt after clicking the checker (unchecked)
Steps to test	1. Just click the checker button to uncheck
Data inputs	Nothing to be inputed
Expected Results	A toast message “(Person Name) has been found. Thank you for giving a helping hand” will be displayed
Actual Results	Meet the expected results
Test Results	Pass

Test Case ID	TC-B-001
Objective	Test the TaskRegistrationActivity.kt with no selection of image
Steps to test	1. Don’t select anything
Data inputs	Nothing to be inputed
Expected Results	A toast message “Please Select an image” will be displayed
Actual Results	Meet the expected results
Test Results	Pass

Test Case ID	TC-B-002
Objective	Test the TaskRegistrationActivity.kt with no input of Person Name
Steps to test	1. Don’t type anything
Data inputs	Nothing to be inputed
Expected Results	A toast message “Please enter both Person Name and Describe the details of the missing persona” will be displayed
Actual Results	Meet the expected results
Test Results	Pass

Test Case ID	TC-B-003
Objective	Test the TaskRegistrationActivity.kt with no input of Describe the details of the missing person
Steps to test	1. Don’t type anything
Data inputs	Nothing to be inputed
Expected Results	A toast message “Please enter both Person Name and Describe the details of the missing person” will be displayed
Actual Results	Meet the expected results
Test Results	Pass

Test Case ID	TC-B-004
Objective	Test the TaskRegistrationActivity.kt with no selection of Date
Steps to test	1. Don’t choose any date
Data inputs	Nothing to be inputed
Expected Results	A toast message “Please select date and time” will be displayed
Actual Results	Meet the expected results
Test Results	Pass

Test Case ID	TC-B-005
Objective	Test the TaskRegistrationActivity.kt with no selection of Time
Steps to test	1. Don’t choose any time
Data inputs	Nothing to be inputed
Expected Results	A toast message “Please select date and time” will be displayed
Actual Results	Meet the expected results
Test Results	Pass

Test Case ID	TC-C-001
Objective	Test the TaskDetailActivity.kt with no new update
Steps to test	1. Remain unchanged for the data input at TaskRegistrationActivity.kt 2. Just click UPDATE button
Data inputs	Nothing to be inputed
Expected Results	A toast message “No change has been found” will be displayed
Actual Results	Meet the expected results
Test Results	Pass

Test Case ID	TC-D-001
Objective	Test after the restart of the app to see whether the previous entries are recorded
Steps to test	1. Add few Missing Person details 2. Restart the app
Data inputs	Person 1: 1. Select image of the missing person 2. Person Name: Chin Zi Ming 3. Describe the details of the missing person: Age:21, Missing Area: Thailand 4. Select Date: 21/7/2023 5. Select Time: 20:54 (pm) Person 2: 1. Select image of the missing person 2. Person Name: Chin Min Liang 3. Describe the details of the missing person: Age:22, Missing Area: Myanmar 4. Select Date: 11/7/2023 5. Select Time: 10:54 (am) Person 3: 1. Select image of the missing person 2. Person Name: Chin Min Lin 3. Describe the details of the missing person: Age:25, Missing Area: Malaysia 4. Select Date: 31/7/2023 5. Select Time: 21:54 (pm)
Expected Results	All these three missing person details are still recorded.
Actual Results	Meet the expected results
Test Results	Pass