**Documentation: 365Robot Telepresence System**

**Overview**

This **Android-powered application** serves as a telepresence system that allows admins to remotely control robots and communicate with users through video calls. Key functionalities include:

* **Robot Control via WebSockets:** The app receives movement commands via WebSockets and executes them on the robot.
* **Real-time Position Updates:** The robot continuously sends its current coordinates to the WebSocket server.
* **WebRTC Video Calls:** The app integrates with the **Agora SDK** for live video and audio communication.
* **Robot Navigation:** Users can store and navigate to predefined positions.
* **Robot SN Management:** The app stores the robot’s **serial number (SN)** and a **distance threshold** in SharedPreferences. (Used to determine how far a point can be and still be considered “close” to the robot)

**Features**

* **WebSocket Communication:** Enables two-way real-time data exchange for commands and updates.
* **Video Call Integration:** Uses **Agora WebRTC SDK** for video conferencing.
* **Secure Settings:** A PIN-protected settings dialog prevents unauthorized changes to the robot’s configuration.

**Code Structure**

**1. MainActivity (Android Application)**

**Description**

Manages the UI and core functionality of the **telepresence system**.

* Loads **Robot SN** and **distance threshold** from SharedPreferences.
* Displays the **settings dialog**.
* Starts **VideoActivity** when the user clicks **Join Now**.

**Implementation**

val sharedPreferences = context.getSharedPreferences(PREFS\_NAME, Context.MODE\_PRIVATE)

val robotSN = sharedPreferences.getString(KEY\_ROBOT\_SN, "") ?: ""

val distanceThresh = sharedPreferences.getString(KEY\_DISTANCE\_THRESH, "0") ?: "0"

* Loads stored values.
* Ensures **Robot SN** is entered before enabling video calls.

**2. WebRTC Video Call Integration**

**Description**

* Uses **AgoraRTC** for live video streaming.
* Allows muting/unmuting audio and enabling/disabling video.
* Notifies the admin when a user joins the call.

**Implementation**

mEngine = initEngine(context, eventHandler, channelName, userRole, videoCallStateViewModel)

* Initializes **AgoraRTC**.
* Joins a **video channel** using the **Robot SN** as the channel name.

**Event Listeners**

override fun onUserJoined(uid: Int, elapsed: Int) {

val notification = buildNotification(context)

sendNotification(context, notification)

}

* Sends a notification when a remote user joins.

**3. WebSocket Communication & Robot Control**

**Description**

* Establishes a **WebSocket connection** for real-time communication.
* Receives movement commands and position updates.

**Implementation**

val request = Request.Builder().url("wss://hungrygowhere.com.sg:8686").build()

webSocket = client.newWebSocket(request, object : WebSocketListener() {

override fun onMessage(webSocket: WebSocket, text: String) {

handleCommand(text)

}

})

* Connects to the **WebSocket server**.
* Listens for **robot commands** and executes movements.

**Handling Commands**

fun handleCommand(command: String) {

when (command) {

"move forward" -> controlRobot(0f, -1f)

"turn left" -> controlRobot(-1f, 0f)

"turn right" -> controlRobot(1f, 0f)

"stop" -> stopMovement()

}

}

* Processes movement commands received via WebSockets.

**Sending Real-Time Position Updates**

RobotApi.getInstance().registerStatusListener(Definition.STATUS\_POSE, object : StatusListener() {

override fun onStatusUpdate(type: String?, value: String?) {

try {

if (value.isNullOrEmpty()) return

val jsonObject = JSONObject(value)

val robotX = jsonObject.getDouble("px")

val robotY = jsonObject.getDouble("py")

jsonObject.put("robotSN", robotSN)

jsonObject.put("command", "pose\_update")

webSocket.send(jsonObject.toString())

} catch (e: JSONException) {

Log.e("onStatusUpdate", "Error parsing status update: ${e.message}")

}

}

})

* The robot's position is updated and sent via WebSocket for real-time tracking.

**Error Handling & Validation**

* **API Failures:** Displays error messages if data retrieval fails.
* **WebSocket Issues:** Logs **invalid WebSocket messages** for debugging.
* **PIN Protection:** Prevents unauthorized changes to **Robot SN**.
* **Invalid Navigation:** Ensures positions are valid before movement.

**Conclusion**

This **Android & WebRTC-powered telepresence system** provides a seamless way to control and interact with robots remotely. By integrating **WebSockets and AgoraRTC**, the system ensures **real-time communication, secure authentication, and precise robot control**, enhancing remote monitoring and control capabilities.