```
class DashboardActivity : AppCompatActivity() {

    private var userToken: String = ""

    private lateinit var bottomNavView: BottomNavigationView
    private lateinit var pagerView: ViewPager2

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

        supportActionBar?.hide()
        userToken = intent.getStringExtra("user_token").orEmpty()

        Log.e("Success", userToken)

        initializeViews()
        configureViewPager()
        configureBottomNavigation()
    }

    private fun initializeViews() {
        bottomNavView = findViewById(R.id.bottom_navigation)
        pagerView = findViewById(R.id.view_pager_container)
    }

    private fun configureViewPager() {
        pagerView.adapter = ViewPagerAdapter(this).apply {
            addFragment(SensorDataFragment())
            addFragment(ModuleStateFragment())
            addFragment(TechnicalCardFragment())
            addFragment(UserAccountFragment())
        }

        pagerView.registerOnPageChangeCallback(object : ViewPager2.OnPageChangeCallback()
{
            override fun onPageSelected(position: Int) {
                bottomNavView.selectedItemId = when (position) {
                    0 -> R.id.navigation_sensor_data
                    1 -> R.id.navigation_module_state
                    2 -> R.id.navigation_technical_card
                    else -> R.id.navigation_user_account
                }
            }
        })
    }

    private fun configureBottomNavigation() {
        bottomNavView.setOnNavigationItemSelectedListener { item ->
            val position = when (item.itemId) {
                R.id.navigation_sensor_data -> 0
                R.id.navigation_module_state -> 1
```

```
            R.id.navigation_technical_card -> 2
            R.id.navigation_user_account -> 3
            else -> return@setOnNavigationItemSelectedListener false
        }
        pagerView.setCurrentItem(position, true)
        true
    }


bottomNavView.findViewById<View>(R.id.navigation_sensor_data).setOnLongClickListener {
        showInfoDialog()
        false
    }
  }

  private fun showInfoDialog() {
    BoxesFragment.newInstance("Box Details").show(supportFragmentManager,
"fragment_show_boxes")
  }

  override fun onBackPressed() {
    with(supportFragmentManager) {
      if (backStackEntryCount > 0) {
        popBackStack()
        bottomNavView.visibility = View.VISIBLE
      } else {
        super.onBackPressed()
      }
    }
  }
}
```

```
@AndroidEntryPoint
class SensorDataFragment : Fragment() {

  private lateinit var requestQueue: RequestQueue
  private lateinit var fragmentLayout: ConstraintLayout

  private val sensorTextViews by lazy {
    mapOf(
      "temperature" to R.id.temperature_value,
      "humidity" to R.id.humidity_value,
      "brightness" to R.id.brightness_value,
      "fluid_level" to R.id.fluid_level_value,
      "soil_moisture" to R.id.soil_moisture_value
    ).mapValues { view?.findViewById<TextView>(it.value) }
  }

  private val sensorSwitches by lazy {
```

```
        mapOf(
            "temperature_control" to R.id.temperature_control_switch,
            "humidity_control" to R.id.humidity_control_switch,
            "brightness_control" to R.id.brightness_control_switch,
            "soil_control" to R.id.soil_control_switch
        ).mapValues { view?.findViewById<Switch>(it.value) }
    }

    private val manualControls by lazy {
        mapOf(
            "temperature_manual" to Pair(R.id.manual_control_temperature,
TemperatureFragment()),
            "humidity_manual" to Pair(R.id.manual_control_humidity, HumidityFragment()),
            "brightness_manual" to Pair(R.id.manual_control_brightness, BrightnessFragment()),
            "soil_manual" to Pair(R.id.manual_control_soil, SoilFragment())
        ).mapValues { view?.findViewById<ImageView>(it.value.first) to it.value.second }
    }

    private var userToken: String = ""
    private var currentBoxId: String = ""
    private var currentBoxTitle: String = ""

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        userToken = requireActivity().intent.getStringExtra("user_token").orEmpty()

        ViewModelProvider(requireActivity()).get(BoxViewModel::class.java).data.observe(this) {
box ->
            updateBoxDetails(box)
            fetchSensorDataFromServer(userToken, currentBoxId)
        }
    }

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? = inflater.inflate(R.layout.sensor_data_fragment, container, false)

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        fragmentLayout = requireActivity().findViewById(R.id.fragment_container)
        fragmentLayout.foreground?.alpha = 0
        requestQueue = Volley.newRequestQueue(requireContext())

        updateHeader()
        fetchSensorDataFromServer(userToken, getBoxDetails("id"))
        initializeSwitches()
        initializeManualControls()
    }

    private fun updateBoxDetails(box: Box) {
        currentBoxId = box.boxId
```

```
            currentBoxTitle = box.boxTitle
            requireActivity().getSharedPreferences("box_details",
        Context.MODE_PRIVATE).edit().apply {
                putString("id", currentBoxId)
                putString("title", currentBoxTitle)
                apply()
            }
        }

        private fun updateHeader() {
            val headerView = view?.findViewById<TextView>(R.id.sensor_data_header)
            headerView?.text = getString(R.string.sensor_header_title, getBoxDetails("title"))
        }

        private fun initializeSwitches() {
            sensorSwitches.forEach { (tag, switch) ->
                switch?.isChecked = getSwitchStatus(tag)
                switch?.setOnCheckedChangeListener { _, isChecked ->
                    saveSwitchStatus(tag, isChecked)
                    manualControls[tag]?.first?.visibility = if (isChecked) View.VISIBLE else
        View.GONE
                }
            }
        }

        private fun initializeManualControls() {
            manualControls.forEach { (tag, pair) ->
                pair.first?.setOnClickListener {
                    val fragment = pair.second
                    fragment?.let {
                        requireActivity().supportFragmentManager.beginTransaction()
                            .replace(R.id.fragment_container, it)
                            .addToBackStack(null)
                            .commit()
                    }
                }
            }
        }

        private fun fetchSensorDataFromServer(token: String, boxId: String) {
            val url = "${BuildConfig.SERVER_URL}/sensors/get/$boxId"
            val request = JsonObjectRequest(
                Request.Method.GET, url, null,
                { response ->
                    val sensorsData = response.getJSONObject("sensorsData")
                    sensorTextViews.forEach { (key, textView) ->
                        textView?.text = when (key) {
                            "temperature" -> getString(
                                R.string.temperature_value, sensorsData.getDouble("temperature").toString()
                            )
                            "humidity" -> getString(
                                R.string.humidity_value, sensorsData.getDouble("humidity").toString()
```

```
                    )
                    "brightness" -> getString(
                        R.string.brightness_value, sensorsData.getInt("brightness").toString()
                    )
                    "fluid_level" -> getString(
                        R.string.fluid_level_value, sensorsData.getInt("fluidLevel").toString()
                    )
                    "soil_moisture" -> getString(
                        R.string.soil_moisture_value, sensorsData.getInt("soilMoisture").toString()
                    )
                    else -> ""
                }
            }
        },
        { error -> error.printStackTrace() }
    ) {
        mapOf(
            "Content-Type" to "application/json; charset=UTF-8",
            "Authorization" to "Bearer $token"
        )
    }
    requestQueue.add(request)
}

private fun saveSwitchStatus(tag: String, isChecked: Boolean) {
    requireActivity().getSharedPreferences("sensor_data_state", Context.MODE_PRIVATE)
        .edit().putBoolean(tag, isChecked).apply()
}

private fun getSwitchStatus(tag: String): Boolean {
    return requireActivity().getSharedPreferences("sensor_data_state",
Context.MODE_PRIVATE)
        .getBoolean(tag, false)
}

private fun getBoxDetails(tag: String): String {
    return requireActivity().getSharedPreferences("box_details", Context.MODE_PRIVATE)
        .getString(tag, "").orEmpty()
}
}
```

```
private inner class ToggleStateListener(
    private val toggleSwitch: Switch,
    private val controlImage: ImageView,
    private val identifier: String
) : CompoundButton.OnCheckedChangeListener {

    override fun onCheckedChanged(button: CompoundButton?, isToggled: Boolean) {
```

```
        persistToggleState(identifier, isToggled)

            controlImage.visibility = if (isToggled) {
            showActivationPopup(toggleSwitch, controlImage)
            View.VISIBLE
        } else {
            View.GONE
        }
    }

     private fun persistToggleState(key: String, state: Boolean) {
        requireActivity()
            .getSharedPreferences("toggle_states", Context.MODE_PRIVATE)
            .edit()
            .putBoolean(key, state)
            .apply()
    }
}
```

```
  private inner class ControlActivationClickListener(
    private val controlImage: ImageView,
    private val targetFragment: Fragment,
    private val identifier: String
) : View.OnClickListener {

    override fun onClick(view: View?) {
        openTargetFragment(targetFragment, identifier)
    }
    private fun openTargetFragment(fragment: Fragment, tag: String) {
        requireActivity().supportFragmentManager.beginTransaction()
            .add(R.id.main_container, fragment)
            .addToBackStack(tag)
            .commit()
    }
}
```

```
class DeviceControlFragment : Fragment() {

    private lateinit var requestQueue: RequestQueue
    private lateinit var stateRequestQueue: RequestQueue

    private lateinit var fanToggle: Switch
    private lateinit var pumpToggle: Switch
    private lateinit var lampToggle: Switch

    private lateinit var fanIcon: ImageView
    private lateinit var pumpIcon: ImageView
```

```
private lateinit var lampIcon: ImageView

private lateinit var headerText: TextView

private var deviceId: String = ""
private var deviceName: String = ""

override fun onCreateView(
    inflater: LayoutInflater, container: ViewGroup?,
    savedInstanceState: Bundle?
): View? {
    return inflater.inflate(R.layout.device_control_fragment, container, false)
}

override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)

    requestQueue = Volley.newRequestQueue(requireContext())
    stateRequestQueue = Volley.newRequestQueue(requireContext())

    val authToken = requireActivity().intent.getStringExtra("token").orEmpty()

    headerText = view.findViewById(R.id.device_control_header)

    fanToggle = view.findViewById(R.id.toggle_fan)
    pumpToggle = view.findViewById(R.id.toggle_pump)
    lampToggle = view.findViewById(R.id.toggle_lamp)

    fanIcon = view.findViewById(R.id.icon_fan)
    pumpIcon = view.findViewById(R.id.icon_pump)
    lampIcon = view.findViewById(R.id.icon_lamp)

    deviceId = getDeviceState("id")
    deviceName = getDeviceState("name")

    headerText.text = getString(R.string.device_header_text, deviceName)

    configureToggles(authToken)
    fetchToggleStates(authToken, deviceId)
}

private fun configureToggles(authToken: String) {
    fanToggle.setOnCheckedChangeListener(
        ToggleChangeListener(
            context = requireContext(),
            toggleSwitch = fanToggle,
            iconView = fanIcon,
            iconEnabled = R.drawable.ic_fan_active,
            iconDisabled = R.drawable.ic_fan_inactive,
            token = authToken
        )
    )
```

```
        pumpToggle.setOnCheckedChangeListener(
            ToggleChangeListener(
                context = requireContext(),
                toggleSwitch = pumpToggle,
                iconView = pumpIcon,
                iconEnabled = R.drawable.ic_pump_active,
                iconDisabled = R.drawable.ic_pump_inactive,
                token = authToken
            )
        )

        lampToggle.setOnCheckedChangeListener(
            ToggleChangeListener(
                context = requireContext(),
                toggleSwitch = lampToggle,
                iconView = lampIcon,
                iconEnabled = R.drawable.ic_lamp_active,
                iconDisabled = R.drawable.ic_lamp_inactive,
                token = authToken
            )
        )
    }

    private fun fetchToggleStates(authToken: String, deviceId: String) {
        val url = "${BuildConfig.SERVER_URL}/devices/get/$deviceId"
        val request = JsonObjectRequest(
            Request.Method.GET, url, null,
            { response ->
                val deviceStates = response.getJSONObject("deviceStates")
                fanToggle.isChecked = deviceStates.getBoolean("fan")
                pumpToggle.isChecked = deviceStates.getBoolean("pump")
                lampToggle.isChecked = deviceStates.getBoolean("lamp")
            },
            { error -> error.printStackTrace() }
        ) {
            mapOf(
                "Content-Type" to "application/json; charset=UTF-8",
                "Authorization" to "Bearer $authToken"
            )
        }
        stateRequestQueue.add(request)
    }

    private fun getDeviceState(key: String): String {
        val preferences = requireActivity().getSharedPreferences("deviceStateData",
Context.MODE_PRIVATE)
        return preferences.getString(key, "").orEmpty()
    }
}
```