```kotlin
package net.zeevox.nearow.ui.fragment

import ...

class PerformanceMonitorFragment : Fragment(), DataProcessor.DataUpdateListener {

    private var _binding: FragmentPerformanceTrackerBinding? = null
    private val binding
        get() = _binding!!

    /** for communicating with the service */
    lateinit var mService: DataCollectionService

    /** whether there is an established link with the service */
    private var mBound: Boolean = false

    /** Defines callbacks for service binding, passed to bindService() */
    private val connection =
        object : ServiceConnection {

            override fun onServiceConnected(className: ComponentName, service: IBinder) {
                // We've bound to LocalService, cast the IBinder and get LocalService instance
                val binder = service as DataCollectionService.LocalBinder
                mService = binder.getService()
                mBound = true

                // Listen to callbacks from the service
                mService.setDataUpdateListener(this@PerformanceMonitorFragment)
            }

            override fun onServiceDisconnected(arg0: ComponentName) {
                mBound = false
            }
        }

    private lateinit var viewSessionsButton: MaterialButton
    private lateinit var toolbarSwitchGps: SwitchMaterial

    companion object {
        /** Logcat tag used for debugging */
        private val TAG = PerformanceMonitorFragment::class.java.simpleName
    }

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?,
    ): View {
        _binding = FragmentPerformanceTrackerBinding.inflate(inflater, container, false)
        return binding.root
    }


    /**
     * Called when the view previously created by [onCreateView] has been detached from the
     * fragment.
     */
    override fun onDestroyView() {
        super.onDestroyView()
        _binding = null
    }
```

```kotlin
    /** Called when the Fragment is visible to the user. */
    override fun onStart() {
        Log.i(TAG, "Fragment started")
        super.onStart()
        startAndBindToDataCollectionService()
    }

    /** Called when the Fragment is no longer started. */
    override fun onStop() {
        Log.i(TAG, "Fragment stopped")
        super.onStop()
        requireContext().unbindService(connection)
        mBound = false
    }

    /**
     * Register the permissions callback, which handles the user's response to the system
     * permissions dialog. https://developer.android.com/training/permissions/requesting
     */
    private val requestPermissionLauncher =
        registerForActivityResult(ActivityResultContracts.RequestPermission()) { isGranted: Boolean
            ->
            // if permission has been granted return to where we left off and start the service
            if (isGranted) {
                if (!mBound) startAndBindToDataCollectionService()
            } else {
                // create an alert (dialog) to explain functionality loss since permission has been
                // denied
                val permissionDeniedDialog =
                    this.let {
                        val builder = AlertDialog.Builder(requireContext())
                        builder.apply {
                            setTitle(getString(R.string.dialog_title_gps_permission_denied))
                            setMessage(getString(R.string.dialog_msg_gps_permission_denied))
                              setPositiveButton(android.R.string.ok) { dialog, _ -> dialog.dismiss()
}
                        }
                        // Create the AlertDialog
                        builder.create()
                    }

                // show the dialog itself
                permissionDeniedDialog.show()
            }
        }

    private fun openSessionsHistory() =
        startActivity(Intent(requireActivity(), SessionsActivity::class.java))
```

```
/**
 * Called immediately after [onCreateView] has returned, but before any saved state has been
 * restored in to the view.
 */
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)

    // do not let the screen dim or switch off
    binding.root.keepScreenOn = true

    // TODO create alternative landscape layout file to support rotation
    requireActivity().requestedOrientation = ActivityInfo.SCREEN_ORIENTATION_PORTRAIT

    toolbarSwitchGps = binding.pmToolbar.findViewById(R.id.pm_toolbar_switch_gps)
    viewSessionsButton = binding.pmToolbar.findViewById(R.id.pm_toolbar_action_session_history)

    toolbarSwitchGps.setOnCheckedChangeListener { _, isChecked ->
        if (isChecked) mService.enableGps() else mService.disableGps()
        val visibility = if (isChecked) View.VISIBLE else View.GONE
        binding.splitFrame.visibility = visibility
        binding.distanceFrame.visibility = visibility
    }

    viewSessionsButton.setOnClickListener { openSessionsHistory() }

    binding.startStopButton.setOnClickListener {
        if (!mService.dataProcessor.isRecording) startTracking()
        else Snackbar.make(
                    binding.root,
                    getString(R.string.info_use_long_press_to_stop),
                    Snackbar.LENGTH_LONG)
                .show()
    }

    // long click used to stop recording to prevent water splashes from stopping session
    binding.startStopButton.setOnLongClickListener {
        if (!mService.dataProcessor.isRecording) startTracking() else stopTracking()
        true
    }
}

/**
 * Called when stroke rate is recalculated [strokeRate]
 * - estimated rate in strokes per minute
 */
override fun onStrokeRateUpdate(strokeRate: Double) {
    binding.strokeRate.text = String.format("%.1f", strokeRate)
}

/** Called when a new GPS fix is obtained */
override fun onLocationUpdate(location: Location, totalDistance: Float) {
    binding.apply {
        splitFrame.visibility = View.VISIBLE
        distanceFrame.visibility = View.VISIBLE
        split.text = UnitConverter.speedToSplitFormatted(location.speed)
        distance.text = String.format("%.0f", totalDistance)
    }
}
```

```
    private fun startTracking() {
        // reset chronometer
        binding.timer.base = SystemClock.elapsedRealtime()
        // start counting up
        binding.timer.start()
        mService.dataProcessor.startRecording()
        binding.startStopButton.apply {
            text = getString(R.string.action_stop_tracking)
            backgroundTintList =
                ColorStateList.valueOf(ResourcesCompat.getColor(resources, R.color.end_red, null))
            // https://stackoverflow.com/a/29146895
            icon = ResourcesCompat.getDrawable(resources, R.drawable.ic_round_stop_24, null)
        }

        viewSessionsButton.isEnabled = false
        toolbarSwitchGps.isEnabled = false
    }

    private fun stopTracking() {
        binding.timer.stop()
        mService.dataProcessor.stopRecording()
        binding.startStopButton.apply {
            text = getString(R.string.action_start_tracking)
            backgroundTintList =
                ColorStateList.valueOf(
                    ResourcesCompat.getColor(resources, R.color.start_green, null))
            icon = ResourcesCompat.getDrawable(resources, R.drawable.ic_round_play_arrow_24, null)
        }

        viewSessionsButton.isEnabled = true
        toolbarSwitchGps.isEnabled = true
    }

    /** Bind to LocalService. If it is not running, automatically start it up */
    private fun startAndBindToDataCollectionService() {
        if (ActivityCompat.checkSelfPermission(
            requireContext(), Manifest.permission.ACCESS_FINE_LOCATION) !=
            PackageManager.PERMISSION_GRANTED) {
            // request GPS permission, callback will re-call this function to start the service
            requestPermissionLauncher.launch(Manifest.permission.ACCESS_FINE_LOCATION)
            return
        }

        val dataCollectionServiceIntent =
            Intent(requireContext(), DataCollectionService::class.java)

        // mark the service as started so that it is not killed
        // https://stackoverflow.com/a/43742797
        // the startService and startForegroundService methods can be called
        // as many times as necessary -- there will always only be one
        // instance of the service running
        // https://developer.android.com/guide/components/services#StartingAService
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O)
            requireContext().startForegroundService(dataCollectionServiceIntent)
        else requireContext().startService(dataCollectionServiceIntent)

        // bind and automatically create the service
        requireContext()
            .bindService(dataCollectionServiceIntent, connection, Context.BIND_AUTO_CREATE)
    }
}
```