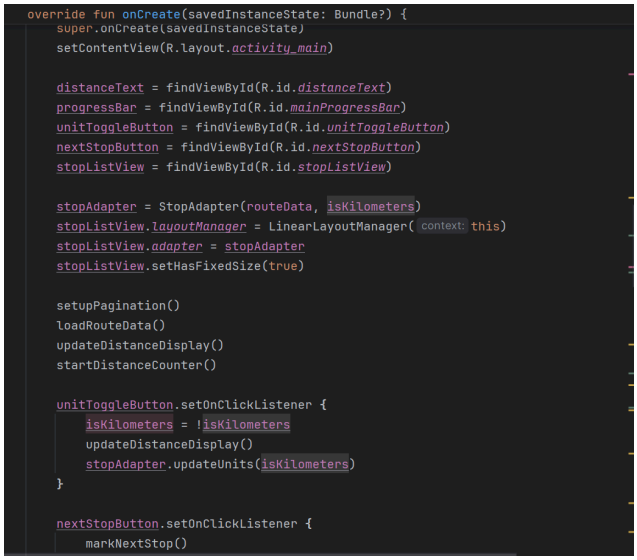
Assignment 1

Roll No: **2021235**
Arnav Agarwal

MainActivity (Main Logic)

```
private fun setupPagination() {
    stopListView.addOnScrollListener(object : OnScrollListener() {
        override fun onScrolled(recyclerView: RecyclerView, dx: Int, dy: Int) {
            val layoutManager = recyclerView.layoutManager as LinearLayoutManager
            val lastVisibleItemPosition = layoutManager.findLastVisibleItemPosition(

            if (lastVisibleItemPosition == layoutManager.itemCount - 1) {
                loadMoreData()
            }
        }
    })
}

private fun loadMoreData() {
    val newStops = mutableListOf<RouteStop>()

    for (i in 0 ≤ until < 3) {
        if (fileLineIndex >= fileData.size) break

        val line = fileData[fileLineIndex]
        val parts = line.split( ...delimiters: "-")

        if (parts.size == 3) {
            val stopName = parts[0].split( ...delimiters: ":")[1].trim()
            val distance = parts[1].trim().toIntOrNull() ?: 0
```

- **File:** MainActivity.kt
- **Description:**
  - Manages the overall flow of the app, including loading data, updating UI, and tracking trip progress.
  - **Features:**
    - Loads route and distance data from a file (seoul_to_sau_paulo.txt).
    - Displays total trip progress with a progress bar.
    - Supports toggling between kilometers and miles.
    - Allows users to mark stops as reached, updating both the list and progress bar.
    - Also manages the lazy loading/pagination functionality with a maximum of 3 items loaded.
    - The `startDistanceCounter` method in `MainActivity.kt` increments distance covered every second.

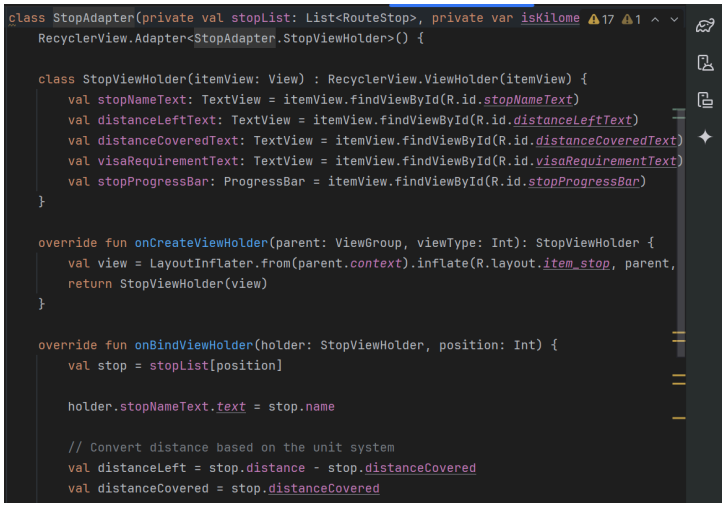


```kotlin
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    distanceText = findViewById(R.id.distanceText)
    progressBar = findViewById(R.id.mainProgressBar)
    unitToggleButton = findViewById(R.id.unitToggleButton)
    nextStopButton = findViewById(R.id.nextStopButton)
    stopListView = findViewById(R.id.stopListView)

    stopAdapter = StopAdapter(routeData, isKilometers)
    stopListView.layoutManager = LinearLayoutManager( context: this)
    stopListView.adapter = stopAdapter
    stopListView.setHasFixedSize(true)

    setupPagination()
    loadRouteData()
    updateDistanceDisplay()
    startDistanceCounter()

    unitToggleButton.setOnClickListener {
        isKilometers = !isKilometers
        updateDistanceDisplay()
        stopAdapter.updateUnits(isKilometers)
    }

    nextStopButton.setOnClickListener {
        markNextStop()
```

Route Data Management

- **File:** `RouteStop.kt`
- **Description:**
  - Defines the `RouteStop` data class, which represents a travel stop.
  - **Fields:**
    - `name`: Name of the stop.
    - `distance`: Distance to the next stop.
    - `visaRequirement`: Visa requirement for the stop.
    - `distanceCovered`: Distance covered dynamically updated during the trip.

RecyclerView (Stop List Display)

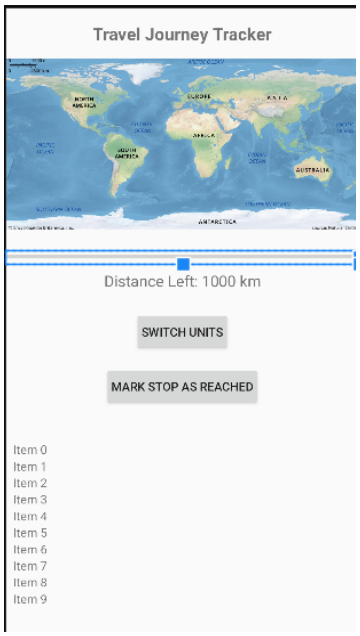


```kotlin
class StopAdapter(private val stopList: List<RouteStop>, private var isKilome 17 1
    RecyclerView.Adapter<StopAdapter.StopViewHolder>() {

    class StopViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
        val stopNameText: TextView = itemView.findViewById(R.id.stopNameText)
        val distanceLeftText: TextView = itemView.findViewById(R.id.distanceLeftText)
        val distanceCoveredText: TextView = itemView.findViewById(R.id.distanceCoveredText)
        val visaRequirementText: TextView = itemView.findViewById(R.id.visaRequirementText)
        val stopProgressBar: ProgressBar = itemView.findViewById(R.id.stopProgressBar)
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): StopViewHolder {
        val view = LayoutInflater.from(parent.context).inflate(R.layout.item_stop, parent,
        return StopViewHolder(view)
    }

    override fun onBindViewHolder(holder: StopViewHolder, position: Int) {
        val stop = stopList[position]

        holder.stopNameText.text = stop.name

        // Convert distance based on the unit system
        val distanceLeft = stop.distance - stop.distanceCovered
        val distanceCovered = stop.distanceCovered
```

- **File:** `StopAdapter.kt`

- **Description:**
    - Adapter for displaying a list of stops in the RecyclerView.
    - **Features:**
        - Binds stop data to the layout in `item_stop.xml`.
        - Updates the UI to show distance left, distance covered, and visa requirements.
        - Dynamically updates each stop's progress bar based on the covered distance.

XML Layouts



Main Activity Layout

- **File:** `activity_main.xml`
- **Description:**
    - Contains the main UI components:
        - Title, World Map image, overall progress bar, distance display, and buttons.
        - RecyclerView for displaying travel stops.

Stop Item Layout

- **File:** `item_stop.xml`
- **Description:**
    - Layout for each stop item in the list.
    - Includes:
        - Stop name, distance left, distance covered, visa requirement, and a progress bar for individual stop progress.

Compose Version

MainActivity (Entry Point)

- File: `MainActivity.kt`
- Description:
  - The `MainActivity` sets the content for the app using Compose. It initializes the TravelJourneyTrackerApp composable function.

TravelJourneyTrackerApp

- **Composable:** `TravelJourneyTrackerApp()`
- **Description:**
  - This function builds the main UI structure of the app using **Scaffold**.
  - The **topBar** displays the title and world map image.
- **Components:**
  - **Text:** Displays the app title.
  - **Image:** Displays a world map image from the drawable resource.

```
@Composable
fun TravelJourneyTrackerApp() {
    Scaffold(
        topBar = {
            Column(
                horizontalAlignment = Alignment.CenterHorizontally,
                modifier = Modifier
                    .fillMaxWidth()
                    .padding(16.dp)
            ) {
                Text(text = "Travel Journey Tracker", fontSize = 20.sp)

                // Add the Image composable for the map
                Image(
                    painter = painterResource(id = R.drawable.world_map),
                    contentDescription = "World Map",
                    modifier = Modifier
                        .fillMaxWidth()
                        .height(200.dp)
                        .padding(top = 16.dp)
                )
            }
        }
    ) { contentPadding ->
        MainContent(contentPadding)
    }
}
```

MainContent

```
@Composable
fun MainContent(contentPadding: PaddingValues) {
    val context = LocalContext.current
    var isKilometers by remember { mutableStateOf( value: true) }
    val stops = remember { loadStopsFromFile(context) }
    var currentStopIndex by remember { mutableStateOf( value: 0) }
    var totalDistanceLeft by remember { mutableStateOf(stops.sumOf { it.distance }) }

    // LazyListState to monitor scroll position
    val lazyListState = rememberLazyListState()

    // Function to handle moving to the next stop
    fun markNextStop() {
        if (currentStopIndex < stops.size) {
            val currentStop = stops[currentStopIndex]
            currentStop.distanceCovered = currentStop.distance  // Set distance covered
            totalDistanceLeft = stops.sumOf { it.distance - it.distanceCovered }  // Up
            currentStopIndex++  // Move to the next stop
        }
    }
```

```
    // Lazy List to Display Stops (limited to 3 visible items at a time)
    LazyColumn(
        modifier = Modifier.fillMaxSize(),
        state = lazyListState
    ) {
        items(stops) { stop ->
            StopItem(stop = stop, isKilometers = isKilometers)
        }
    }
}
```

```
fun StopItem(stop: RouteStop, isKilometers: Boolean) {
    Card(
        modifier = Modifier
            .fillMaxWidth()
            .padding(8.dp)
    ) {
        Column(modifier = Modifier.padding(16.dp)) {
            Text(text = "Stop Name: ${stop.name}", fontSize = 16.sp)

            // Display "Reached" when stop is fully covered
            if (stop.distanceCovered >= stop.distance) {
                Text(
                    text = "Reached",
                    fontSize = 14.sp,
                    color = Color.Green
                )
            } else {
                Text(
                    text = "Distance Left: ${
                        if (isKilometers) stop.distance - stop.distanceCovered
                        else String.format("%.2f miles", (stop.distance - stop.distanc
                    }",
                    fontSize = 14.sp
                )
            }
```

- **Composable:** `MainContent(contentPadding: PaddingValues)`
- **Description:**
    - Handles the core logic and UI for tracking travel progress.
    - **Components:**
        - Distance display, progress bar, buttons, and stop list (LazyColumn).

- **Functionality**
    1. **State Variables:**
        - `isKilometers`: Toggles between kilometers and miles.
        - `stops`: Loads route stops from a file.
        - `currentStopIndex` and `totalDistanceLeft`: Track the current stop and remaining distance.
    2. **markNextStop Function:**
        - Marks the current stop as reached and updates the remaining distance.
    3. **Distance Counter:**
        - Uses **LaunchedEffect** to simulate distance being covered over time.

StopItem

- **Composable:** `StopItem(stop: RouteStop, isKilometers: Boolean)`
- **Description:**
    - Displays details for each travel stop.
    - **Components:**
        - Stop name, distance left, visa requirement, and individual progress bar.