Mobile Application Development

Week9　Add Location and Map to Android App

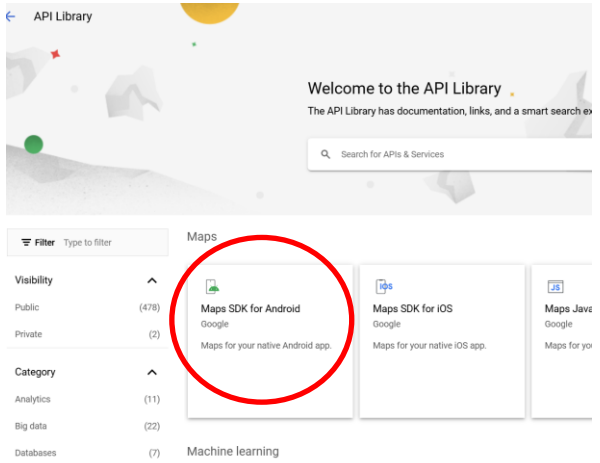
Yi SUN
Kobe Institute of Computing

Add the map for the Shopping App

1. Open the ShoppingApp Project
2. Add the navigation dependencies
3. Prepare the UI for Navigation
4. Set up the Routes
5. Implement the Navigation and pass the data
6. Serialization and Deserialization with Parceable

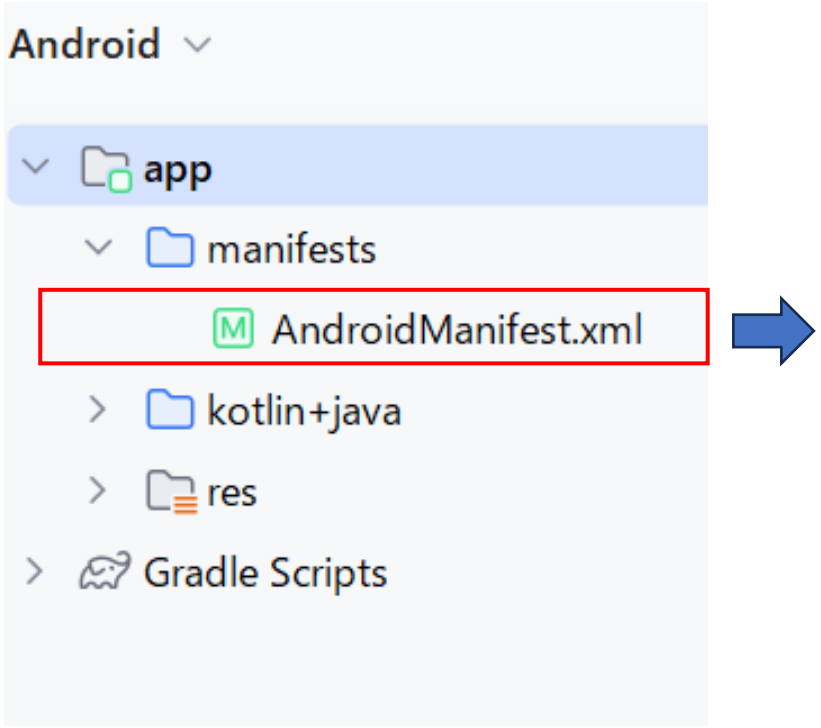
Set up the Google Cloud API Key for Map service

- Access and login https://cloud.google.com
- Access to the Console screen, and add a new project
- Access to the APIs & services, and click +Enable APIs and servies
- Click the Maps SDK for Android and Enable Map service API
- Copy the API Key standby for use.



Add Permissons for Location Serves in AndroidManifest.xml

- Edit the AndroidManifest.xml file in app/manifests/ folder



AndroidManifest.xml

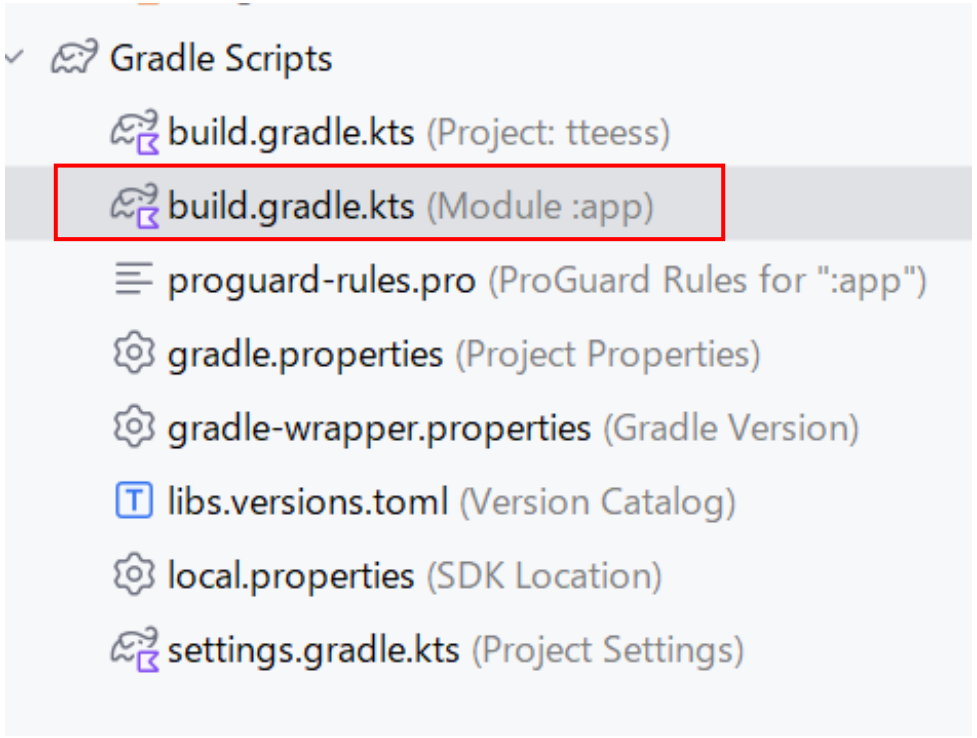
```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

    <application
        android:allowBackup="true"
…
        android:theme="@style/Theme.ShoppingApp"
        tools:targetApi="31">

        <meta-data
            android:name="com.google.android.geo.API_KEY"
            android:value="AIzaSyA2_TSre2tKmKMu3Wl02U4e0hFz71kQwXs"
            />
        <activity
            android:name=".MainActivity"
```

The google Map API Key

https://developer.android.com/reference/android/Manifest.permission

Add the Dependency

- Make sure you've added the correct dependency in your build.gradle (app-level) file:




```
dependencies {

    val nav_version = "2.8.8"
    implementation("androidx.navigation:navigation-compose:$nav_version")
    implementation("androidx.lifecycle:lifecycle-viewmodel-compose:2.8.7")
    implementation("com.squareup.retrofit2:retrofit:2.11.0")
    implementation("com.squareup.retrofit2:converter-gson:2.11.0")
    implementation("io.coil-kt.coil3:coil-compose:3.1.0")
    implementation("io.coil-kt.coil3:coil-network-okhttp:3.1.0")
    implementation("com.google.android.gms:play-services-location:21.3.0")
    implementation("com.google.maps.android:maps-compose:6.4.1")
    implementation(libs.androidx.core.ktx)
    implementation(libs.androidx.lifecycle.runtime.ktx)
```

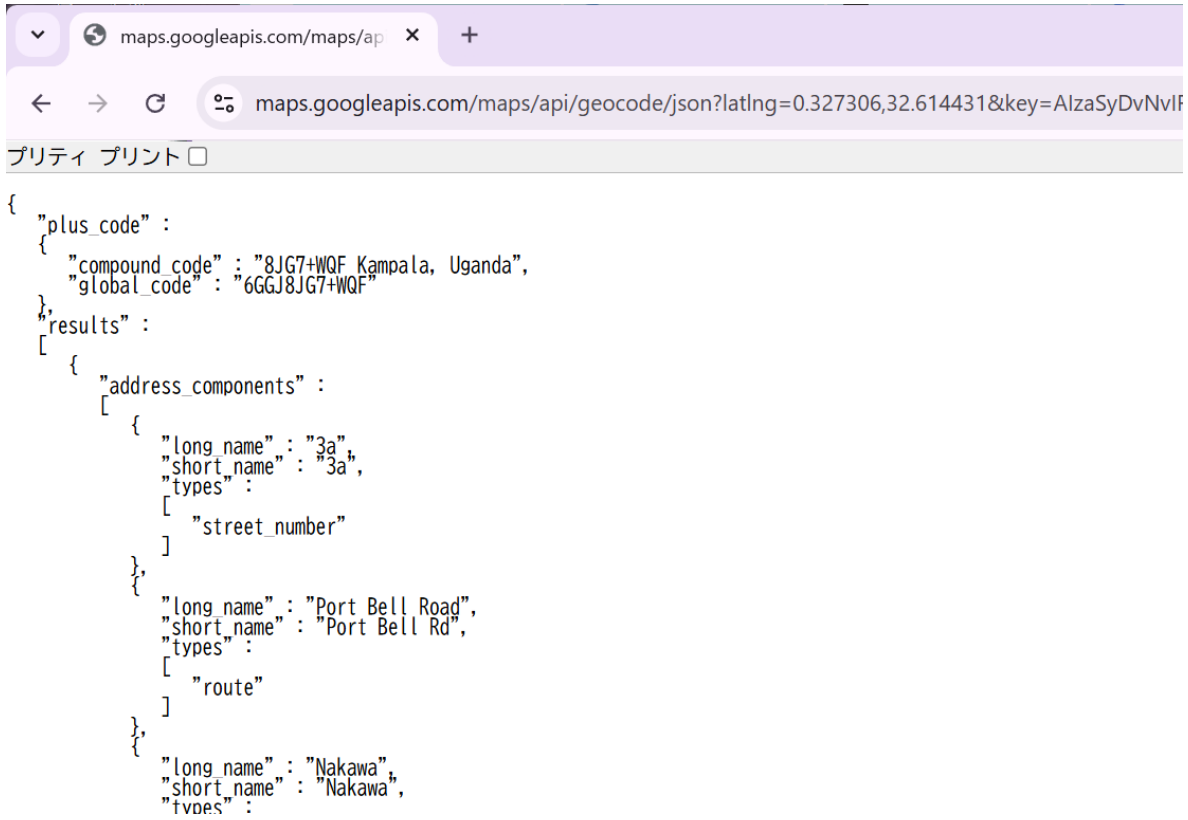
https://developers.google.com/android/guides/setup

https://developers.google.com/maps/documentation/android-sdk/maps-compose

Test to access API from web

- Access the Map API service from web browser

https://maps.googleapis.com/maps/api/geocode/json?latlng=0.327306,32.614431&key=“your API Key”



Latitude

Longitude

https://developers.google.com/maps/documentation/geocoding/start

Add a Data class for location data

Create a New data class file **LocationData** to store the location data and the Goecoding response.

LocationData.kt

```
data class LocationData(
    val latitude: Double,
    val longitude: Double,
    val address: String = ""
)

data class GeocodingResponse(
    val results: List<GeocodingResult>,
    val status: String
)

data class GeocodingResult(
    val formatted_address: String
)
```

Add the Location Selection Screen

Add a new file ***LocationSelectionScreen.kt***

LocationSelectionScreen.kt

```
@Composable
fun LocationSelectionScreen(
    location: LocationData,
    onLocationSelected: (LocationData) -> Unit
){
    val userLocation = remember {
        mutableStateOf(LatLng(location.latitude,location.longitude))
    }

    var cameraPositionState = rememberCameraPositionState{
        position = CameraPosition.fromLatLngZoom(userLocation.value,10f)
    }
}
```

Add the Google Map to Screen

```
fun LocationSelectionScreen(
    location: LocationData,
    onLocationSelected: (LocationData) -> Unit
){
    val userLocation = remember {
        mutableStateOf(LatLng(location.latitude,location.longitude))
    }
    var cameraPositionState = rememberCameraPositionState{
        position = CameraPosition.fromLatLngZoom(userLocation.value,10f)
    }
    Column(modifier = Modifier.fillMaxSize()) {
        GoogleMap(
            modifier = Modifier.weight(1f).padding(top=16.dp),
            cameraPositionState = cameraPositionState,
            onMapClick = {
                userLocation.value = it
            }
        ){
            Marker(state = rememberMarkerState(position = userLocation.value))
        }
        var newLocation:LocationData

        Button(onClick = {
            newLocation = LocationData(userLocation.value.latitude,userLocation.value.longitude)
            onLocationSelected(newLocation)
        }) {
            Text("Set Location")
        }
    }
}
```

Created using rememberCameraPositionState and initialized with a CameraPosition centered at userLocation.value and a zoom level of 10. This state controls the view (position and zoom) of the Google Map.

GoogleMap Component:
Placed inside the Column with modifiers to give it weight (it takes most of the space) and a top padding.
cameraPositionState: Passed to control the current view of the map.
onMapClick:A lambda that gets triggered when the map is clicked. It updates the userLocation state with the clicked position, meaning the marker on the map will move to where the user tapped.
Marker:Within the map, a Marker is added. It uses rememberMarkerState with the current userLocation to indicate the selected position visually on the map.

When the button is clicked, it creates a new LocationData object using the current latitude and longitude from userLocation and then calls the onLocationSelected callback with this new location. This confirms the user's selection.

https://developers.google.com/maps/documentation/android-sdk/maps-compose

Add the location view model

LocationViewModel.kt

```
class LocationViewModel: ViewModel(){
    private val _location = mutableStateOf<LocationData?>(null)
    val location: State<LocationData?> = _location

    fun updateLocation(newLocation: LocationData){
        _location.value = newLocation
    }
}
```

Add the location data to MainViewModel

MainViewModel.kt

```
class MainViewModel:ViewModel() {
    private val _productsState = mutableStateOf(ProductState())
    val productState: State<ProductState> = _productsState

    private val _selectedLocation = mutableStateOf<LocationData?>(null)
    val selectedLocation: State<LocationData?> = _selectedLocation

    init {
        fetchProducts()
    }

    fun updateSelectedLocation(location: LocationData) {
        _selectedLocation.value = location
    }

    private fun fetchProducts(){
        viewModelScope.launch {
            try {
```

Add a Button at ProductDetail Screen

ProductDetailScreen.kt

```
fun ProductDetailScreen(
    product: Product,
    deliveryLocation: LocationData?,
    onSelectLocation: ()-> Unit
    ){
    Column(modifier = Modifier
…
    )
    {
        Text(text=product.title, textAlign = TextAlign.Center)
        Image(
            painter = rememberAsyncImagePainter(product.image),
…
        )
         Text(text=product.description,
…
        )
        Spacer(modifier = Modifier.height(16.dp))
        Button(onClick = onSelectLocation ) {
            Text(text = "Select Delivery Location" )
        }
        Text(
            text = "Delivery Address:",
            modifier = Modifier.padding(top = 16.dp),
            textAlign = TextAlign.Center
        )
```

A nullable LocationData that represents the current delivery location

A callback function that gets invoked when the user wants to choose or change the delivery location.

Triggers the onSelectLocation callback when clicked.

Regisiter the location screen at route

Screen.kt

```
sealed class Screen(val route:String) {
    object  ShoppingScreen:Screen("main_screen")
    object  ItemDetailScreen:Screen("detail_screen")
    object  LocationSelectionScreen:Screen("location_screen")
}
```

Modify the navigation

ShoppingApp.kt

```
fun ShoppingApp(navController: NavHostController, modifier: Modifier){
    val productViewModel: MainViewModel = viewModel()
    val viewstate by productViewModel.productState
    val selectedLocation by productViewModel.selectedLocation

    NavHost(navController=navController, startDestination = Screen.ShoppingScreen.route){
        composable(route=Screen.ShoppingScreen.route){
            ShoppingScreen(viewstate=viewstate, navigationToDetail = {
.....}
        composable(route=Screen.ItemDetailScreen.route){
            val product = navController.previousBackStackEntry?.savedStateHandle?.
            get<Product>("item")?: Product(0,"",0.0,"","")
            ProductDetailScreen(
                product=product,
                deliveryLocation = selectedLocation,
                onSelectLocation = {
                    navController.navigate(Screen.LocationSelectionScreen.route)
                }
....
        composable(route=Screen.LocationSelectionScreen.route) {
            LocationSelectionScreen(
                location = selectedLocation ?: LocationData(0.0,0.0),
                onLocationSelected = {location ->
                    productViewModel.updateSelectedLocation(location)
                    navController.popBackStack()
                }
```

Observes the current selected location from the ViewModel.

Displays detailed information about the selected product.
Shows the current deliveryLocation.
Provides a callback (onSelectLocation) that navigates to the LocationSelectionScreen when the user wants to select or change the delivery location.

Displays a map or interface to pick a location.
Uses the current selectedLocation (or a default location if none is set).
Callback onLocationSelected:
When a new location is selected, it updates the ViewModel's state by calling updateSelectedLocation.
Then, it navigates back (pops the back stack) to return to the previous screen (likely the item detail screen).

Finalizing the Location Selection Screen

LocationSelectionScreen.kt

```
fun LocationSelectionScreen(
    location: LocationData,
    onLocationSelected: (LocationData) -> Unit
) {
    val context = LocalContext.current
    val locationUtils = remember { LocationUtils(context) }
    val locationViewModel = remember { LocationViewModel() }

    // Default to Kampala coordinates if no location is available
    val defaultLocation = LatLng(0.3476, 32.5825) // Kampala coordinates

    var userLocation by remember {
        mutableStateOf(defaultLocation)
    }

    var currentGpsLocation by remember {
        mutableStateOf<LatLng?>(null)
    }

    var isLoading by remember { mutableStateOf(true) }
    var currentAddress by remember { mutableStateOf("") }
    var hasReceivedLocation by remember { mutableStateOf(false) }
    var showPermissionDialog by remember { mutableStateOf(false) }

    val markerState = rememberMarkerState(position = userLocation)
    val cameraPositionState = rememberCameraPositionState {
        position = CameraPosition.fromLatLngZoom(userLocation, 15f)
    }
```

Context and Utility Setup

Prepare the State Variables and Default Values

Finalizing the Location Selection Screen

LocationSelectionScreen.kt

```
// Permission launcher
val permissionLauncher = rememberLauncherForActivityResult(
    contract = ActivityResultContracts.RequestMultiplePermissions()
) { permissions ->
    val locationPermissionsGranted = permissions.entries.all { it.value }
    if (locationPermissionsGranted) {
        locationUtils.requestLocationUpdates(locationViewModel)
    } else {
        // Use fallback location if permissions denied
        val fallbackLocation = if (location.latitude != 0.0 || location.longitude != 0.0) {
            LatLng(location.latitude, location.longitude)
        } else {
            defaultLocation
        }
        userLocation = fallbackLocation
        markerState.position = fallbackLocation
        cameraPositionState.position = CameraPosition.fromLatLngZoom(fallbackLocation, 15f)
        isLoading = false
    }
}
// Try to get current location first
LaunchedEffect(Unit) {
    if (locationUtils.hasLocationPermission(context)) {
        locationUtils.requestLocationUpdates(locationViewModel)
    } else {
        showPermissionDialog = true
    }
}
```

Permission Launcher:
Uses rememberLauncherForActivityResult to request location permissions at runtime:

A LaunchedEffect runs on composition: This checks for permissions and either requests location updates or shows the permission dialog.

Finalizing the Location Selection Screen

Permission Dialog screen (1/3)

An alert dialog that appears when the app needs location permission but it hasn't been granted yet. It uses an if statement to check if showPermissionDialog is true, and if so, displays an AlertDialog with several behaviors:

```
if (showPermissionDialog) {
    AlertDialog(
        onDismissRequest = {
            showPermissionDialog = false
            // Use fallback location if dialog dismissed
            val fallbackLocation = if (location.latitude != 0.0 || location.longitude != 0.0) {
                LatLng(location.latitude, location.longitude)
            } else {
                defaultLocation
            }
            userLocation = fallbackLocation
            markerState.position = fallbackLocation
            cameraPositionState.position = CameraPosition.fromLatLngZoom(fallbackLocation, 15f)
            isLoading = false
        },
```

Sets showPermissionDialog to false, hiding the dialog.
Determines a fallback location. It checks if the provided location has non-zero latitude or longitude. If yes, it uses that location; otherwise, it defaults to defaultLocation.
Updates userLocation, the marker's position (markerState.position), and the map's camera position (cameraPositionState) to this fallback location.
Sets isLoading to false, indicating that any loading state (like waiting for permission or location data) is complete.

Finalizing the Location Selection Screen

Permission Dialog screen(2/3)

```
        title = { Text("Location Permission Required") },
        text = { Text("This app needs access to location to show your
current position on the map. Would you like to grant permission?") },
        confirmButton = {
            Button(onClick = {
                showPermissionDialog = false
                permissionLauncher.launch(
                    arrayOf(
                        Manifest.permission.ACCESS_FINE_LOCATION,
                        Manifest.permission.ACCESS_COARSE_LOCATION
                    )
                )
            }) {
                Text("Grant Permission")
            }
        },
```

Title and Text:
The dialog displays a title ("Location Permission Required") and a message explaining that the app needs location access to show the current position on the map.

Confirm Button:When the user taps the "Grant Permission" button:
The dialog is hidden by setting showPermissionDialog to false.
The app requests the necessary location permissions (ACCESS_FINE_LOCATION and ACCESS_COARSE_LOCATION) by launching the permissionLauncher. This will prompt the system permission dialog for the user to grant or deny location access.

Finalizing the Location Selection Screen

Permission Dialog screen (3/3)

```
dismissButton = {
        Button(onClick = {
            showPermissionDialog = false
            // Use fallback location if permission denied
            val fallbackLocation = if (location.latitude != 0.0 || location.longitude != 0.0) {
                LatLng(location.latitude, location.longitude)
            } else {
                defaultLocation
            }
            userLocation = fallbackLocation
            markerState.position = fallbackLocation
            cameraPositionState.position =
CameraPosition.fromLatLngZoom(fallbackLocation, 15f)
            isLoading = false
        }) {
            Text("Not Now")
        }
    }
  )
}
```

Dismiss Button:
When the user taps the "Not Now" button:
The dialog is dismissed similarly by setting showPermissionDialog to false.
It also calculates the fallback location (using the same logic as in onDismissRequest) and updates userLocation, markerState.position, and cameraPositionState accordingly.
Finally, it sets isLoading to false, ending any loading indicators.

Finalizing the Location Selection Screen

Location Updates and Reverse Geocoding

Observing Location Updates: A LaunchedEffect listens for changes in locationViewModel.location.value:

```
// Observe location updates
LaunchedEffect(locationViewModel.location.value) {
    locationViewModel.location.value?.let { newLocation ->
        if (!hasReceivedLocation && (newLocation.latitude != 0.0 || newLocation.longitude != 0.0)) {
            val currentLocation = LatLng(newLocation.latitude, newLocation.longitude)
            userLocation = currentLocation
            currentGpsLocation = currentLocation
            markerState.position = currentLocation
            cameraPositionState.position = CameraPosition.fromLatLngZoom(currentLocation, 15f)
            hasReceivedLocation = true
        } else if (newLocation.latitude != 0.0 || newLocation.longitude != 0.0) {
            // Update current GPS location without moving the marker
            currentGpsLocation = LatLng(newLocation.latitude, newLocation.longitude)
        }
        isLoading = false
    }
}
```

Finalizing the Location Selection Screen

```
// Update address when location changes
LaunchedEffect(userLocation) {
    currentAddress = withContext(Dispatchers.IO) {
        locationUtils.reverseGeocodeLocation(LocationData(userLocation.latitude,
userLocation.longitude))
    }
    isLoading = false
}

// Cleanup location updates when the composable is disposed
DisposableEffect(Unit) {
    onDispose {
        // Any cleanup if needed
    }
}
```

Another LaunchedEffect triggers when userLocation changes. It uses a background coroutine (withContext(Dispatchers.IO)) to perform reverse geocoding:

A cleanup block is provided for when the composable leaves the composition:

Finalizing the Location Selection Screen

```
Column(                                                    UI Layout
    modifier = Modifier.fillMaxSize(),
    horizontalAlignment = Alignment.CenterHorizontally
) {
    if (isLoading) {
        CircularProgressIndicator(modifier = Modifier.padding(16.dp))
    } else {
        GoogleMap(
            modifier = Modifier
                .weight(1f)
                .padding(top = 16.dp),
            cameraPositionState = cameraPositionState,
            onMapClick = { latLng ->
                userLocation = latLng
                markerState.position = latLng
            } UI Layout
        ) {
            Marker(
                state = markerState,
                title = "Selected Location",
                snippet = currentAddress
            )
        }

        Text(
            text = currentAddress,
            modifier = Modifier.padding(16.dp)
        )
```

If isLoading is true, a CircularProgressIndicator is shown. Otherwise, the Google Map and additional UI elements are displayed.

Displays the map with the current camera position and marker. Tapping on the map updates the userLocation:

Finalizing the Location Selection Screen

```
Row(
        modifier = Modifier.padding(bottom = 16.dp),
        verticalAlignment = Alignment.CenterVertically
    ) {
        Button(
            onClick = {
                currentGpsLocation?.let { gpsLocation ->
                    userLocation = gpsLocation
                    markerState.position = gpsLocation
                    cameraPositionState.position = CameraPosition.fromLatLngZoom(gpsLocation, 15f)
                }
            },
            enabled = currentGpsLocation != null && locationUtils.hasLocationPermission(context)
        ) {
            Text("My Location")
        }

        Spacer(modifier = Modifier.width(16.dp))

        Button(
            onClick = {
                onLocationSelected(
                    LocationData(
                        latitude = userLocation.latitude,
                        longitude = userLocation.longitude,
                        address = currentAddress
                    )
                )
            }
        ) {
            Text("Set Location")
        }
    }
  }
}
```

Defines a horizontal row that contains two buttons, "My Location" and "Set Location",

Finalizing the ProductDetail Screen

```
Button(onClick = onSelectLocation) {
    Text(text = if (deliveryLocation == null) "Select Delivery Location" else "Change Delivery Location")
}
```

```
if (deliveryLocation != null) {
    Text(
        text = "Delivery Address:",
        modifier = Modifier.padding(top = 16.dp),
        textAlign = TextAlign.Center
    )
    Text(
        text = deliveryLocation.address.ifEmpty {
            "Lat: ${deliveryLocation.latitude}, Long: ${deliveryLocation.longitude}"
        },
        modifier = Modifier.padding(top = 8.dp),
        textAlign = TextAlign.Center
    )
}
```