Universidad Central del Ecuador

Dispositivos Móviles

Grupo 3

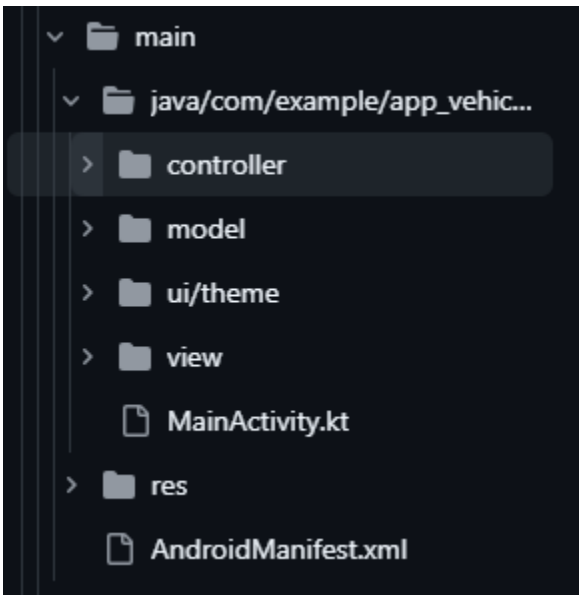
Documentación – Lógica de la aplicación

1. Arquitectura: MVC

Nuestra aplicación hace uso del patrón de arquitectura MVC, lo que hace que cuente con 3 paquetes principales:

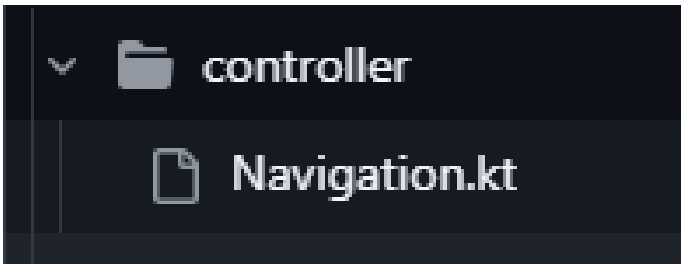
- Controller
- Model
- View

Adicionalmente, cuenta con el paquete Ui/theme (para los estilos). Dentro de todos estos paquetes, se encuentran alojadas las clases que contienen el código fuente.



2. Paquetes y clases

Controller



- ***Navigation.kt***

Es un composable que define la estructura de navegación de la aplicación utilizando NavHost de Jetpack Compose. Gestiona la navegación entre diferentes pantallas (splash screen, login, registro, pantalla principal, añadir vehículo y editar vehículo) y mantiene el estado de las listas de usuarios y vehículos, así como un vehículo seleccionado para edición. Utiliza conceptos de Jetpack Compose como remember, mutableStateListOf, y LaunchedEffect para manejar el estado y efectos secundarios.

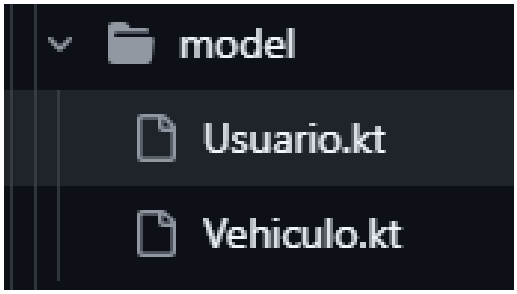
Flujo de Navegación

1. La aplicación inicia en la pantalla de splash, espera 2 segundos y navega al login.
2. Desde el login, el usuario puede:
    - Iniciar sesión y navegar al home.
    - Ir a la pantalla de registro para crear un nuevo usuario.
3. Desde el home, el usuario puede:
    - Cerrar sesión y volver al login.
    - Navegar a addVehiculo para añadir un vehículo.
    - Seleccionar un vehículo para editarlo y navegar a editVehiculo.
    - Eliminar un vehículo directamente.
4. Desde addVehiculo o editVehiculo, el usuario puede guardar los cambios o cancelar, regresando al home.

Composables y Estado

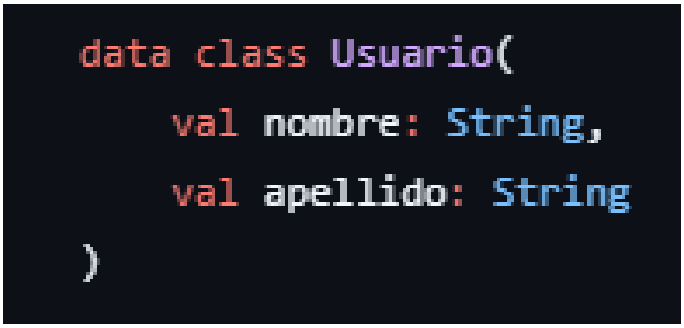
- Uso de remember: Asegura que las listas de usuarios y vehículos, así como el vehículo seleccionado, persistan durante las recomposiciones de la UI.
- Uso de mutableStateListOf: Permite que las listas sean reactivas, es decir, los cambios en las listas (añadir, eliminar, actualizar) provocan que la UI se actualice automáticamente.
- Uso de LaunchedEffect: Ejecuta el retraso y la navegación en la pantalla de splash de forma controlada, evitando múltiples ejecuciones innecesarias.
- Navegación: Utiliza navController.navigate para mover entre pantallas y popBackStack para regresar. La opción popUpTo en la pantalla de splash asegura que no se pueda volver a ella.

Model



- ***Usuario.kt***

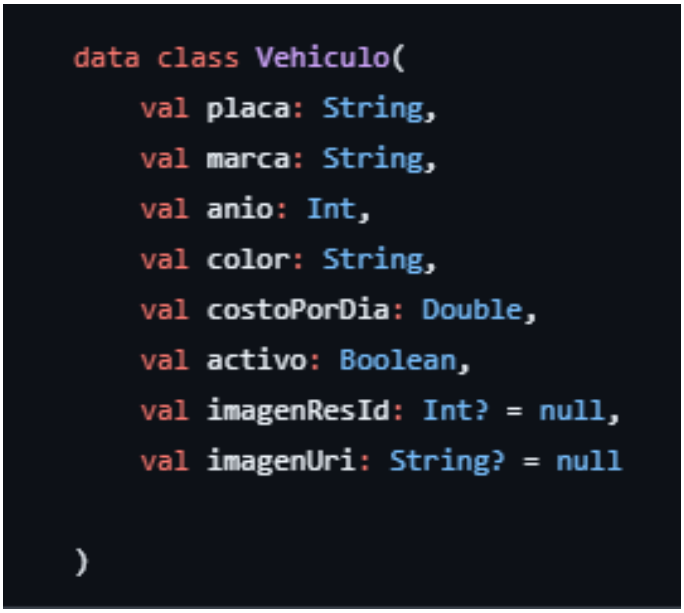
Es una data class que representa un usuario en la aplicación. Contiene dos propiedades: nombre y apellido, ambas de tipo String. Su propósito es almacenar información básica de un usuario.



```
data class Usuario(
    val nombre: String,
    val apellido: String
)
```

- ***Vehiculo.kt***

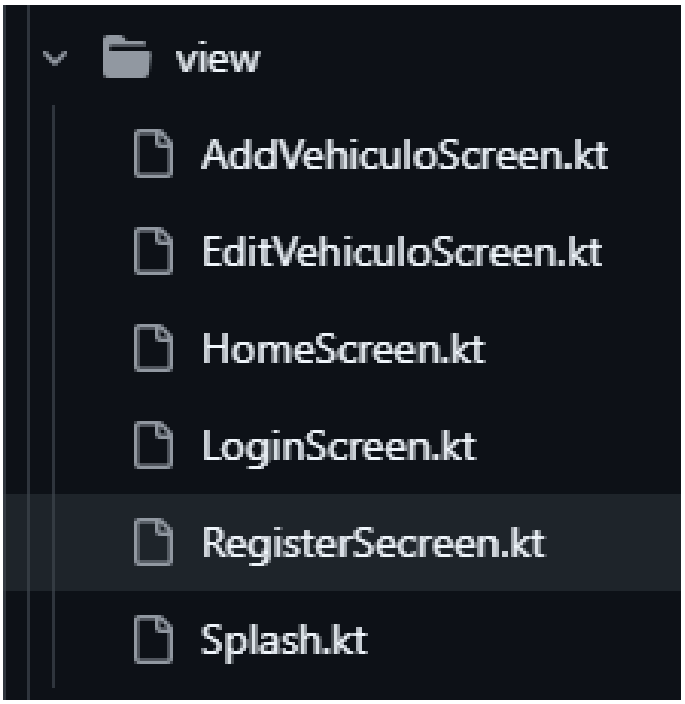
Es una data class que modela un vehículo con propiedades como la placa, marca, año, color, costo por día, estado de actividad y referencias a una imagen (ya sea como un recurso de la aplicación o una URI). Se utiliza para gestionar una lista de vehículos en la aplicación, permitiendo operaciones como mostrar, añadir, editar y eliminar vehículos en pantallas como HomeScreen, AddVehiculoScreen y EditVehiculoScreen.



```
data class Vehiculo(
    val placa: String,
    val marca: String,
    val anio: Int,
    val color: String,
    val costoPorDia: Double,
    val activo: Boolean,
    val imagenResId: Int? = null,
    val imagenUri: String? = null

)
```

View



- ***AddVehiculoScreen.kt***

Es un composable que proporciona una interfaz para añadir o editar un vehículo en la aplicación. Se utiliza en el flujo de navegación para crear un nuevo vehículo o modificar uno existente.

Parámetros:

- **onSave: (Vehiculo) -> Unit:** Callback para guardar el vehículo creado/editado.
- **onCancel: () -> Unit:** Callback para cancelar la operación y regresar.
- **vehiculoAEditar: Vehiculo? = null: V**ehículo opcional para prellenar los campos en modo edición.

Estado:

- **Propiedades Mutables:** Usa mutableStateOf para almacenar placa, marca, anio, color, costoPorDia, activo, imagenUri y showErrors. Si vehiculoAEditar no es nulo, inicializa los campos con sus valores.
- **Validaciones:** Verifica si los campos son válidos (isPlacaValid, isMarcaValid, etc.) para mostrar errores si es necesario.
- **Selección de Imagen:** Usa rememberLauncherForActivityResult con ActivityResultContracts.GetContent para seleccionar una imagen desde la galería (imagenUri).

Interfaz:

- **Diseño:** Un Column con desplazamiento vertical (verticalScroll) que contiene:
    - Un título ("Nuevo Vehículo" o "Editar Vehículo").

    - Campos de texto (OutlinedTextField) para placa, marca, anio, color y costoPorDia, con validaciones y mensajes de error.
    - Un Checkbox para activo.
    - Un botón para seleccionar una imagen desde la galería, que se muestra con Image si se selecciona.
    - Botones "Guardar" y "Cancelar" en un Row.
- **Comportamiento:**
    - **Guardar:** Valida los campos; si son válidos, crea un objeto Vehiculo y llama a onSave. Si no hay imagen seleccionada y no se está editando, usa R.drawable.toyota como imagenResId.
    - **Cancelar:** Llama a onCancel para regresar sin guardar.
    - **Errores:** Muestra mensajes de error si los campos no son válidos al intentar guardar (showErrors = true).

- ***EditVehiculoScreen.kt***

Es un composable de que proporciona una interfaz para editar un vehículo existente en la aplicación.

Parámetros:

- **vehiculo: Vehiculo:** El vehículo a editar, usado para prellenar los campos.
- **onSave: (Vehiculo) -> Unit:** Callback para guardar el vehículo editado.
- **onCancel: () -> Unit:** Callback para cancelar y regresar.

Estado:

- **Propiedades Mutables:** Usa mutableStateOf para placa, marca, anio, color, costoPorDia, activo, imagenUri, imagenSeleccionada (para imágenes precargadas) y expanded (para el menú desplegable). Inicializa los campos con los valores del vehiculo.
- **Selección de Imagen:** Usa rememberLauncherForActivityResult para seleccionar una imagen de la galería (imagenUri). También permite elegir imágenes precargadas (toyota, chevrolet, nissan) si no hay imagenUri.
- **Imágenes Precargadas:** Un mapa (imagenes) asocia nombres con recursos (R.drawable.toyota, etc.), y imagenSeleccionada rastrea la selección.

Interfaz:

- **Diseño:** Un Column con desplazamiento vertical (verticalScroll) que incluye:
    - Título "Editar Vehículo".
    - Campos de texto (OutlinedTextField) para placa, marca, anio, color y costoPorDia.
    - Un Checkbox para activo.
    - Un botón para seleccionar una imagen de la galería.

    - Un ExposedDropdownMenuBox para elegir imágenes precargadas (solo si no hay imagenUri).
    - Una vista previa de la imagen seleccionada (Image) usando rememberAsyncImagePainter para imagenUri o painterResource para imágenes precargadas.
    - Botones "Guardar" y "Cancelar" en un Row.
- **Comportamiento:**
    - **Guardar:** Crea un nuevo objeto Vehiculo con los datos editados y llama a onSave. Usa la imagen precargada si no hay imagenUri, con R.drawable.toyota como predeterminado.
    - **Cancelar:** Llama a onCancel para regresar sin guardar.

- ***HomeScreen.kt***

Es un composable que muestra la pantalla principal de la aplicación, presentando una lista de vehículos y opciones para agregar un vehículo nuevo, editar uno existente o cerrar sesión.

Parámetros:

- **vehiculos: List<Vehiculo>:** Lista de vehículos a mostrar.
- **onLogout: () -> Unit:** Callback para cerrar sesión.
- **onAddVehiculo: () -> Unit:** Callback para navegar a la pantalla de añadir vehículo.
- **onEditVehiculo: (Vehiculo) -> Unit:** Callback para editar un vehículo.
- **onDeleteVehiculo: (Vehiculo) -> Unit:** Callback para eliminar un vehículo.
- **modifier: Modifier = Modifier:** Modificador opcional para personalizar el diseño.

Interfaz:

- **Diseño:** Un Column con:
    - Un Row con un título ("Inicio"), un botón para añadir vehículo y otro para cerrar sesión.
    - Un LazyColumn que muestra una lista de vehículos en tarjetas (Card).
- **Tarjetas de Vehículo:**
    - Cada tarjeta muestra una imagen (de imagenUri con rememberAsyncImagePainter, imagenResId con painterResource, o R.drawable.toyota por defecto).
    - Muestra detalles: placa, marca, anio, color, costoPorDia y activo.
    - Incluye botones "Editar" y "Eliminar" en un Row.
- **Comportamiento:**
    - **Agregar:** Llama a onAddVehiculo para navegar a la pantalla de añadir vehículo.
    - **Cerrar Sesión:** Llama a onLogout para volver al login.

- **Editar:** Llama a onEditVehiculo con el vehículo seleccionado.
- **Eliminar:** Llama a onDeleteVehiculo para eliminar el vehículo de la lista.

- ***LoginScreen.kt***

Es un composable que proporciona una interfaz para que los usuarios inicien sesión en la aplicación verificando sus credenciales (nombre y apellido) contra una lista de usuarios registrados.

Parámetros:

- **usuariosRegistrados: List<Usuario>:** Lista de usuarios registrados para validar las credenciales.
- **onLoginSuccess: () -> Unit:** Callback para navegar a la pantalla principal si el login es exitoso.
- **onGoToRegister: () -> Unit:** Callback para navegar a la pantalla de registro.
- **modifier: Modifier = Modifier:** Modificador opcional para personalizar el diseño.

Estado:

- **Propiedades Mutables:** Usa mutableStateOf para nombre y apellido (como TextFieldValue) y errorMsg (para mensajes de error).
- **Validación:** Busca un usuario en usuariosRegistrados comparando nombre y apellido (sin distinguir mayúsculas/minúsculas).

Interfaz:

- **Diseño:** Un Column centrado con:
  - Título "Login".
  - Campos de texto (OutlinedTextField) para nombre y apellido.
  - Botón "Ingresar" para validar credenciales.
  - Mensaje de error si las credenciales son incorrectas.
  - Botón de texto ("¿No tienes cuenta? Regístrate") para navegar al registro.
- **Comportamiento:**
  - **Ingresar:** Verifica si existe un usuario con el nombre y apellido ingresados. Si coincide, limpia errorMsg y llama a onLoginSuccess. Si no, muestra un mensaje de error.
  - **Registro:** Llama a onGoToRegister para navegar a la pantalla de registro.

- ***RegisterScreen.kt***

Es un composable que proporciona una interfaz para registrar un nuevo usuario en la aplicación, validando que los datos no estén vacíos y que el usuario no exista previamente.

Parámetros:

- **usuarios: List<Usuario>:** Lista de usuarios registrados para verificar duplicados.
- **onRegisterSuccess: (Usuario) -> Unit:** Callback para guardar el nuevo usuario y navegar hacia atrás.
- **modifier: Modifier = Modifier:** Modificador opcional para personalizar el diseño.

Estado:

- **Propiedades Mutables:** Usa mutableStateOf para nombre y apellido (como TextFieldValue) y errorMsg (para mensajes de error).
- **Validaciones:** Comprueba que nombre y apellido no estén vacíos y que no exista un usuario con los mismos datos (ignorando mayúsculas/minúsculas).

Interfaz:

- **Diseño:** Un Column centrado con:
    - Título "Registro".
    - Campos de texto (OutlinedTextField) para nombre y apellido.
    - Botón "Registrarse" para procesar el registro.
    - Mensaje de error si los campos están vacíos o el usuario ya existe.
- **Comportamiento:**
    - **Registrarse:** Valida los campos. Si son válidos y el usuario no está registrado, crea un Usuario y llama a onRegisterSuccess. Si no, muestra un mensaje de error.

- ***Splash.kt***

Es un composable que muestra una pantalla de bienvenida con un logo y un texto animado al iniciar la aplicación.

Estado:

- **Propiedad Mutable:** Usa mutableStateOf para visible (inicialmente false), que controla la animación.
- **Efecto:** Usa LaunchedEffect para establecer visible en true al cargar el composable, activando la animación.

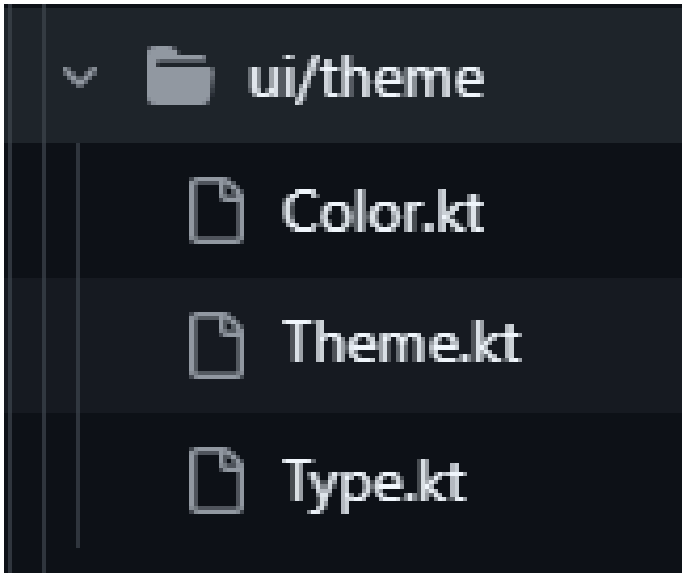
Interfaz:

- **Diseño:** Un Box centrado que contiene un AnimatedVisibility con:
    - Una animación de entrada (fadeIn y scaleIn, duración de 1000 ms, escala inicial de 0.7).
    - Un Column con:
        - Una imagen (Image) del logo (R.drawable.logo2), tamaño 220 dp.
        - Un Spacer de 32 dp.

        - Un texto ("Vehiculos Don Veyker") con estilo personalizado (fuente Pacifico, tamaño 36 sp, color azul).

- **Comportamiento:** Muestra el logo y el texto con una animación de fundido y escalado al cargar.

Ui/theme



Las tres clases trabajan juntas para definir el tema visual de la aplicación:

- **Color.kt** establece una paleta de colores para temas claro y oscuro.
- **Theme.kt** implementa el tema (App_VehiculosTheme), seleccionando esquemas de color (estáticos o dinámicos) y aplicando tipografía.
- **Type.kt** define estilos tipográficos, asegurando consistencia en los textos.

App_VehiculosTheme combina colores y tipografía, aplicándolos a toda la aplicación mediante MaterialTheme. Soporta modo claro/oscuro y colores dinámicos.

3. MainActivity.kt

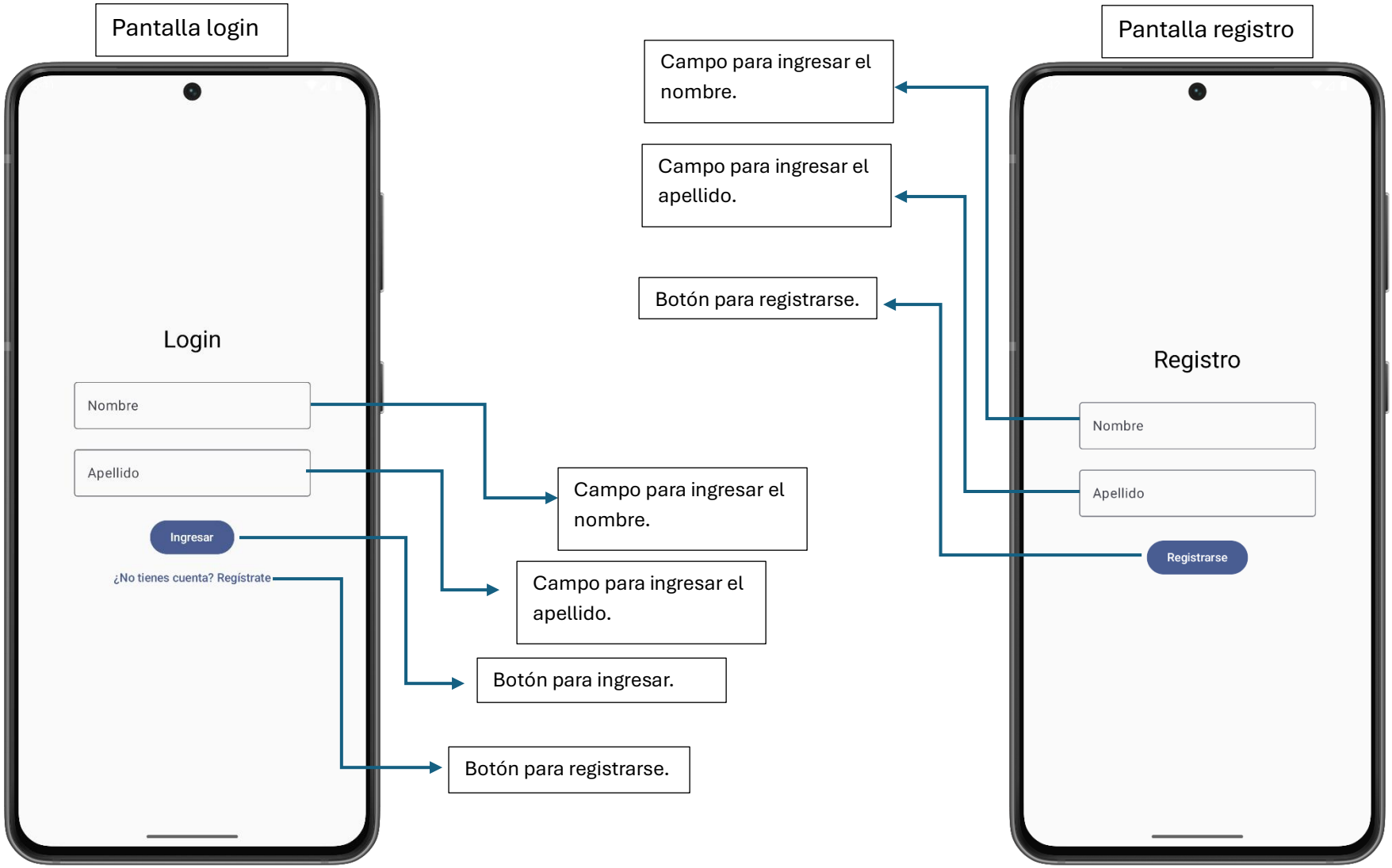
Es la actividad principal de la aplicación Android App_Vehiculos, que configura la interfaz de usuario utilizando Jetpack Compose y gestiona la navegación entre pantallas de manera alternativa a AppNavigation.

Estructura:

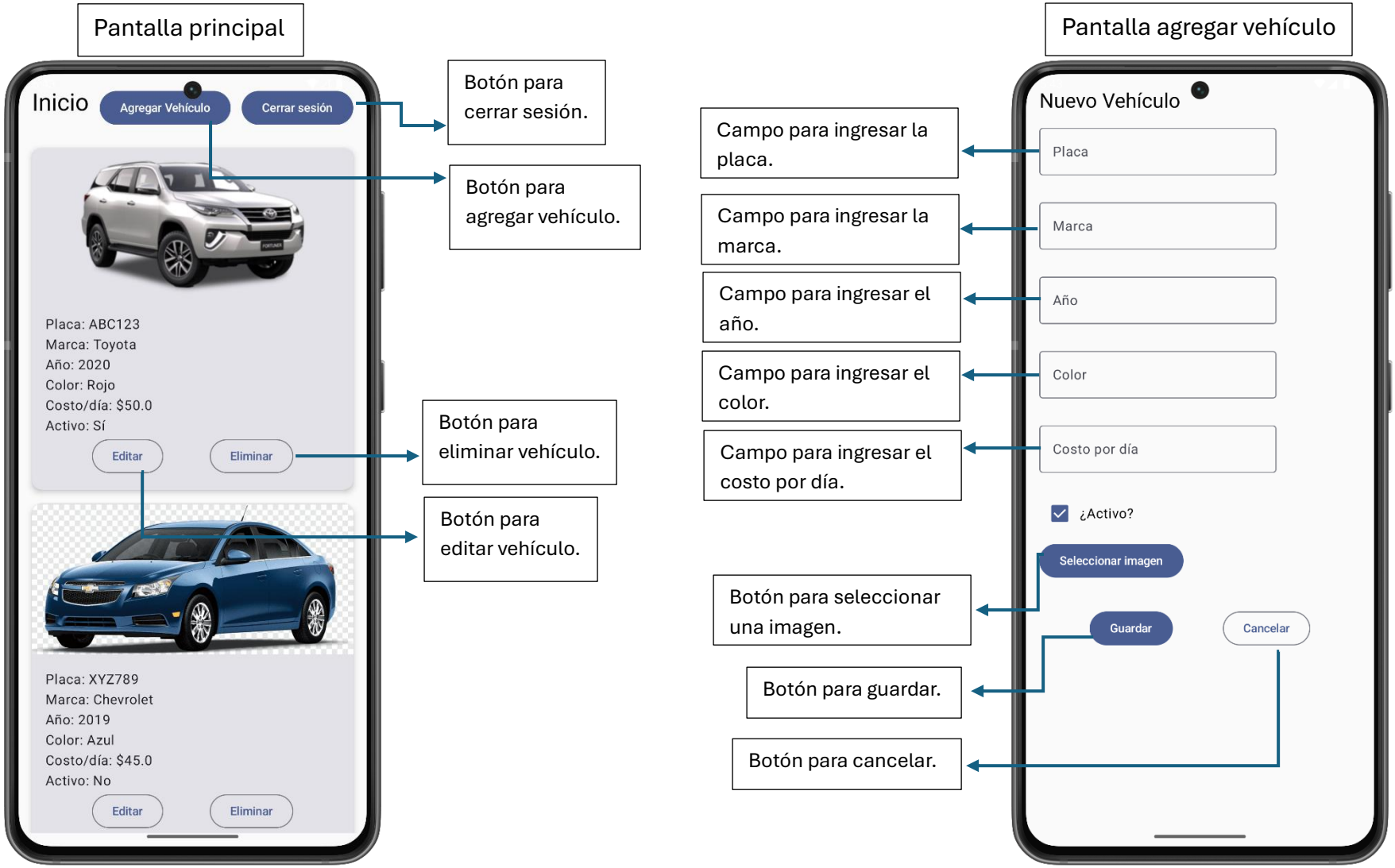
- **Clase MainActivity:**
    - **Propósito:** Extiende ComponentActivity y configura el contenido de la aplicación.
    - **Método onCreate:**

        - Llama a setContent para envolver la aplicación en App_VehiculosTheme (aplicando el tema definido en ui.theme) y renderiza AppNavigation como el composable principal.
    - **Uso:** Actúa como punto de entrada de la aplicación, inicializando el tema y la navegación.
- **Composable App:**
    - **Propósito:** Gestiona la navegación entre pantallas (login, register, home, addVehiculo) utilizando un estado (currentScreen) en lugar de NavHost.
    - **Estado:**
        - **currentScreen:** Controla la pantalla actual (inicialmente "login").
        - **usuariosRegistrados:** Lista mutable de Usuario con valores iniciales para autenticación/registro.
        - **vehiculos:** Lista mutable de Vehiculo con datos iniciales.
        - **vehiculoEnEdicion:** Almacena el vehículo en edición (o null para añadir uno nuevo).
    - **Interfaz:**
        - Usa un bloque when para renderizar una pantalla según currentScreen:
            - **login:** LoginScreen verifica credenciales y permite ir a registro.
            - **register:** RegisterScreen añade nuevos usuarios a usuariosRegistrados y vuelve a login.
            - **home:** HomeScreen muestra vehículos, permite cerrar sesión, añadir, editar o eliminar vehículos.
            - **addVehiculo:** AddVehiculoScreen crea o edita un vehículo, actualizando vehiculos y regresando a home.
    - **Comportamiento:**
        - **Navegación:** Cambia currentScreen para alternar pantallas.
        - **Gestión de Datos:** Actualiza usuariosRegistrados y vehiculos según las acciones (registro, añadir/editar/eliminar vehículos).
        - **Edición:** Usa vehiculoEnEdicion para pasar un vehículo a AddVehiculoScreen en modo edición.

Documentación – Mapa de navegación



Documentación – Mapa de navegación



Documentación – Mapa de navegación