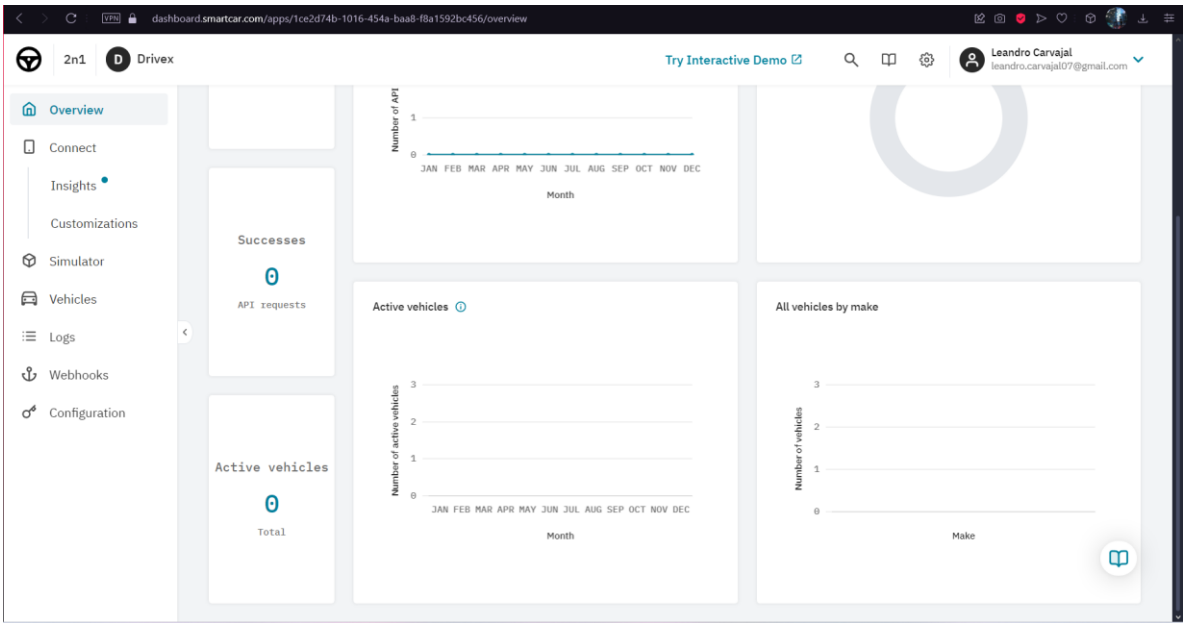
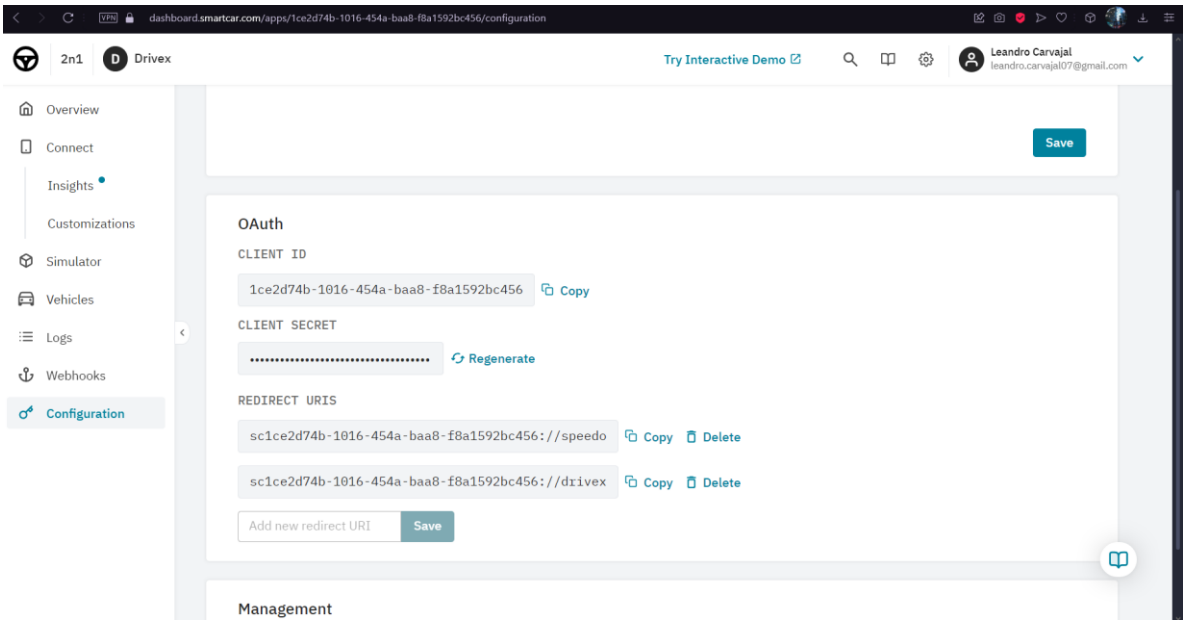
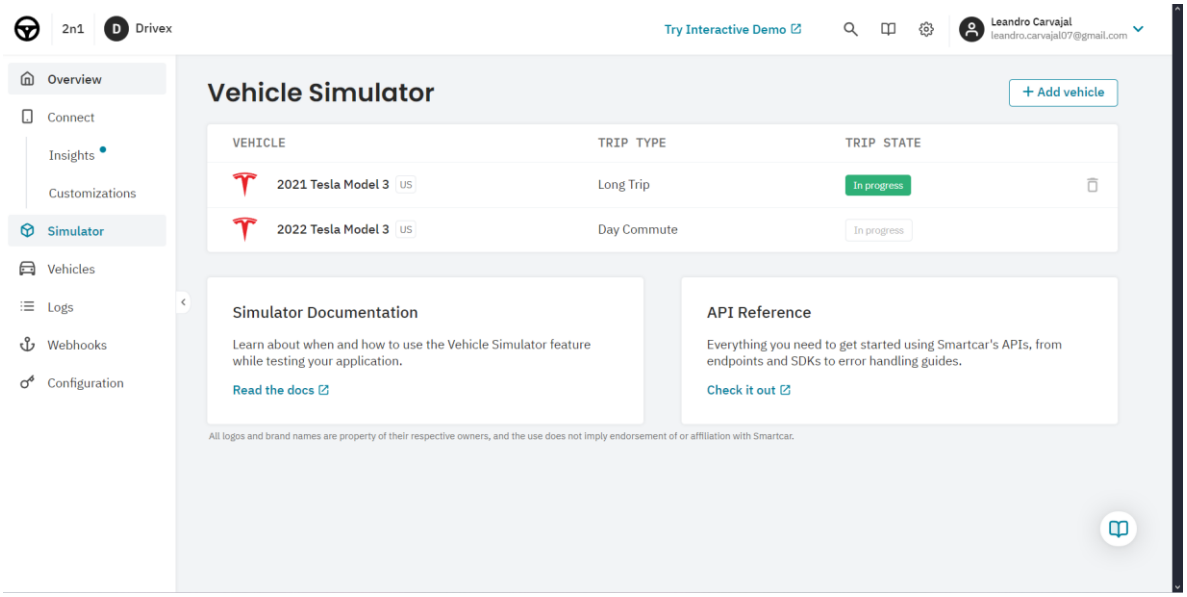
Reporte de pruebas de Smart Car API

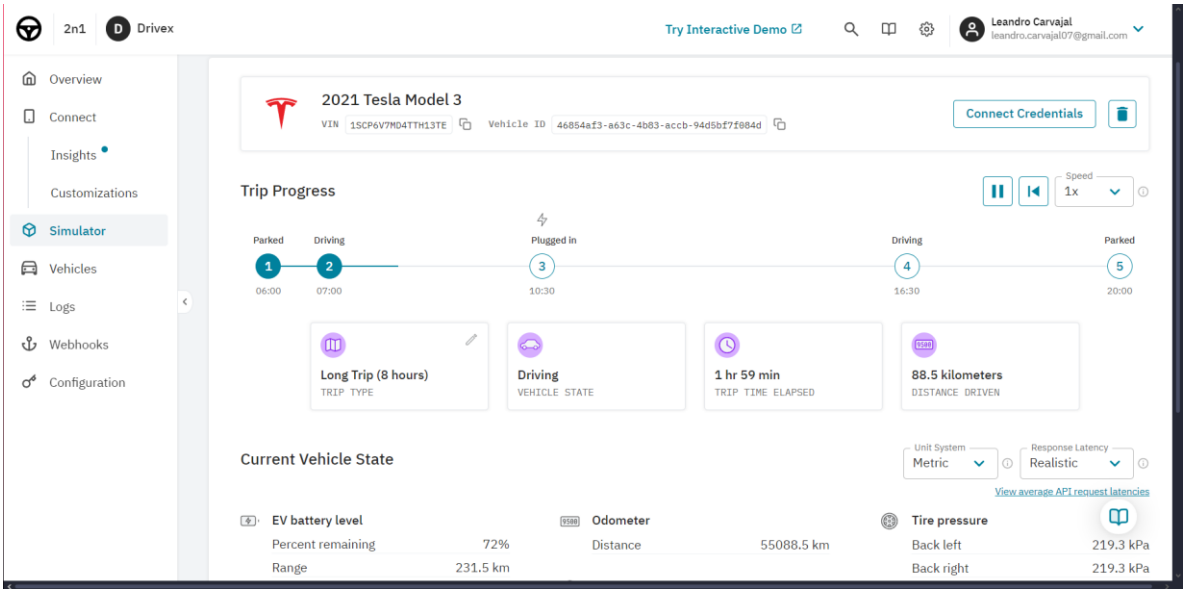
I. Ingreso al dashboard de Smart Car. E información OAuth.

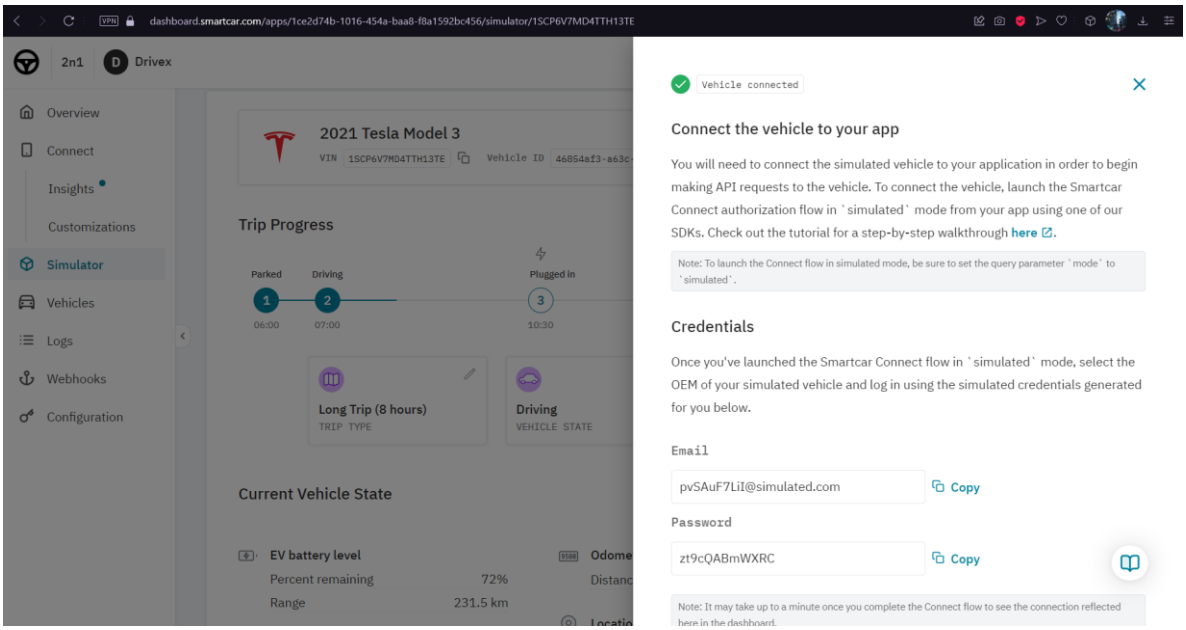




II. Simulación de un vehículo compatible con la API, el cual presenta credenciales simuladas para probar la autentificación.





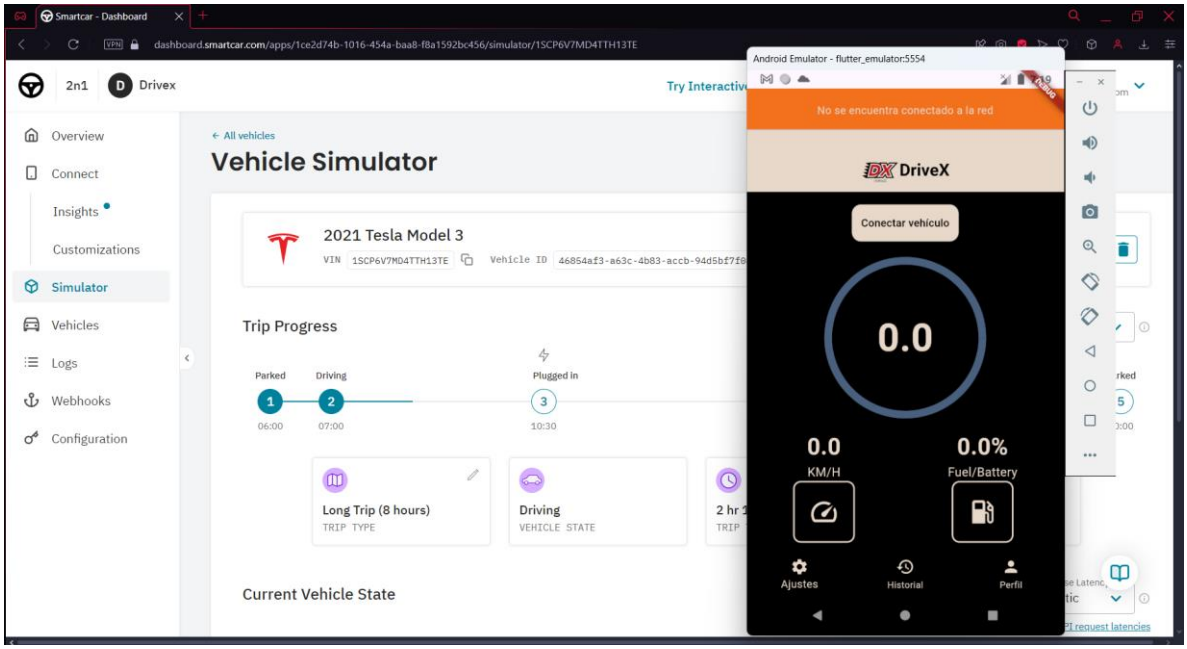


III. Con aquello ya revisado se puede comenzar las pruebas en la aplicación desarrollada en Flutter.

1. Prueba de ingreso a la autenticación Smart Car.

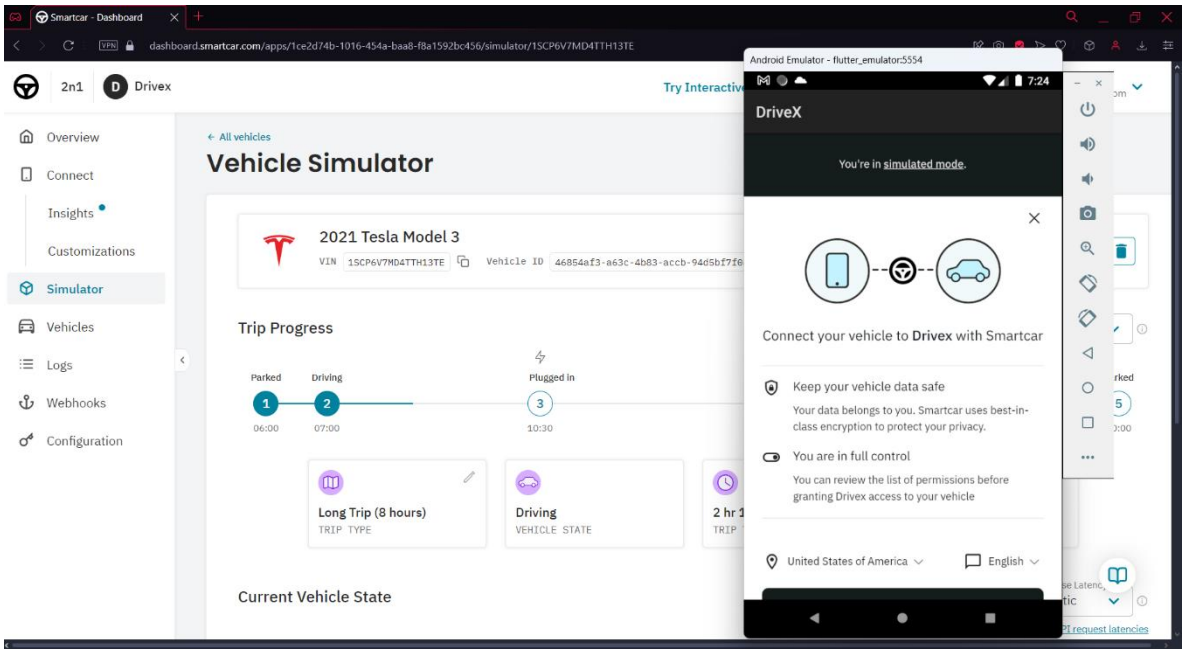
1.1 Sin conexión a internet.

- Resultado esperado: No se inicia el proceso de autentificación.
- Resultado obtenido:



1.2 Con conexión a internet.

- Resultado esperado: Ingreso a la interfaz de autentificación.
- Resultado obtenido:



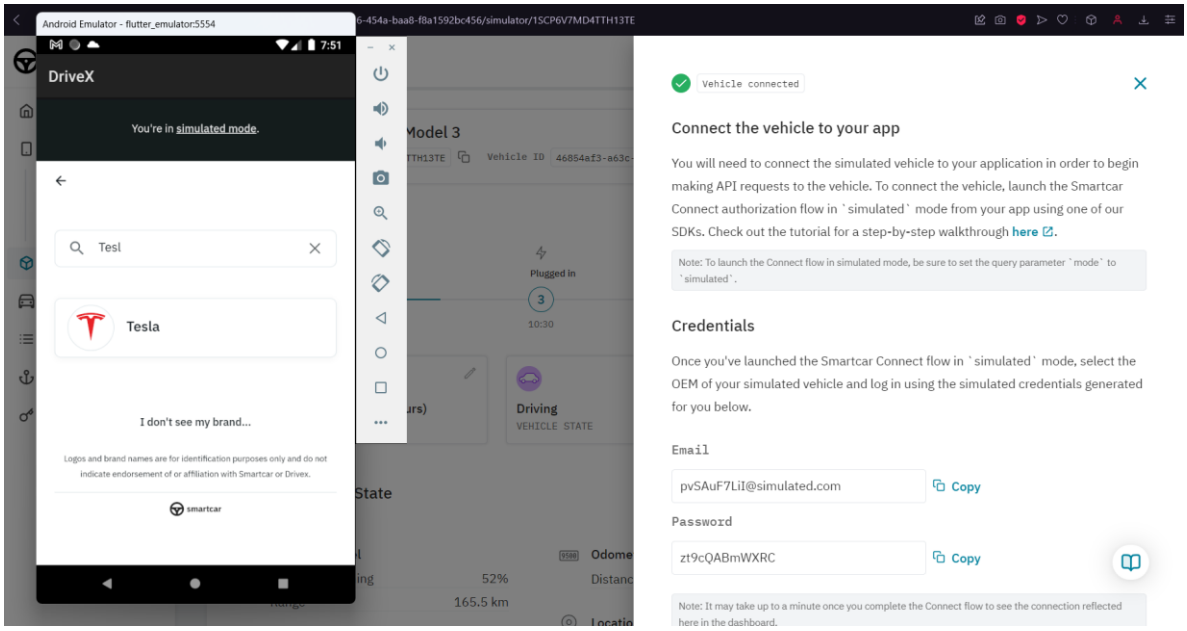
- Código implementado para el requerimiento:

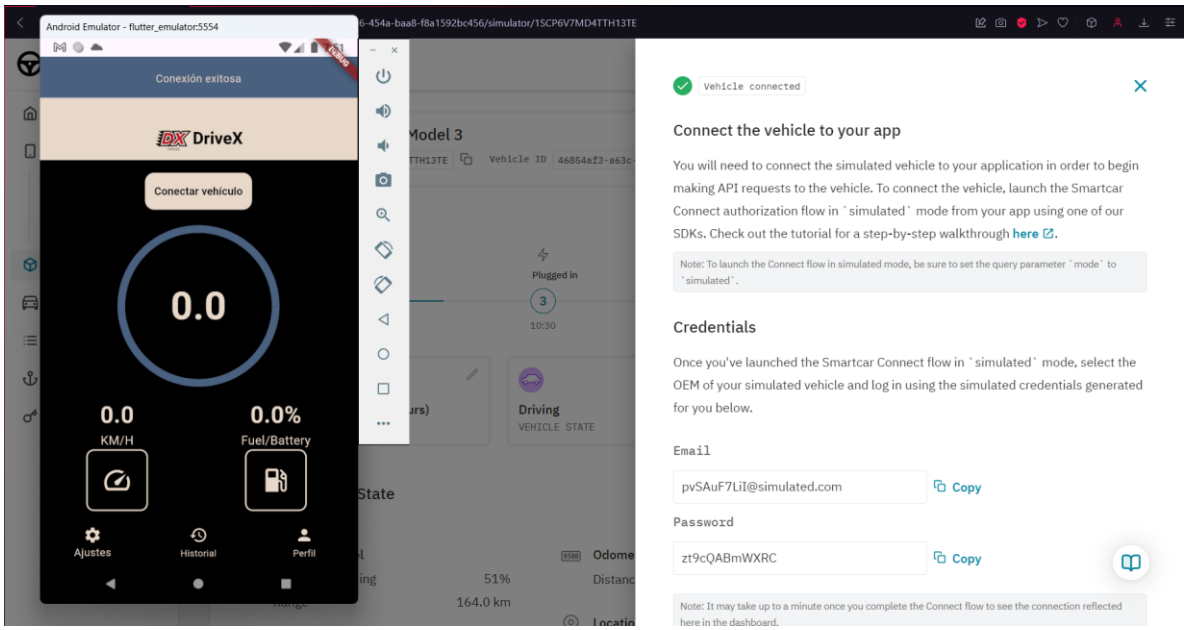
```
Future<void> _connectVehicle() async
  {
    if(APIRequestHelper.instance.isConnected())
    {
      return;
    }

    final List<ConnectivityResult> result = await
Connectivity().checkConnectivity();

    if(result.contains(ConnectivityResult.wifi) ||
result.contains(ConnectivityResult.mobile))
    {
      await Smartcar.launchAuthFlow();
    }
    else
    {
      ScaffoldMessenger.of(context).showMaterialBanner(
          MaterialBanner(
            backgroundColor: GlobalSettings().thirdColor,
            content: Center(
              child: Text(
                'No se encuentra conectado a la red',
                style: TextStyle(
                  color: GlobalSettings().foreColor,
                ),
              ),
            ),
            actions: const [SizedBox.shrink()],
          ),

      );
      Future.delayed(
        const Duration(
          seconds: 3,
        ),
      ).then((_) =>
ScaffoldMessenger.of(context).hideCurrentMaterialBanner());
    }
  }

```

2. Prueba de autentificación de Smart Car

2.1 Ingreso de marca de vehículo y credenciales solicitadas:

- Resultado esperado: Conexión exitosa con el vehículo simulado.
- Resultado obtenido:





- Código implementado para el requerimiento:

```
static Future<void> getAccessToken() async
  {
    Client client = Client();

    var authEncode =
base64.encode(utf8.encode('${APIRequestHelper.instance.clientId}:${APIReques
tHelper.instance.clientSecret}'));
    final Response response = await
client.post(Uri.parse('https://auth.smartcar.com/oauth/token'), headers:
      {
        'Authorization': 'Basic $authEncode',
        'Content-Type': 'application/x-www-form-urlencoded',
      },
      body:
'grant_type=authorization_code&code=${APIRequestHelper.instance.authCode}&re
direct_uri=${APIRequestHelper.instance.uri}'
    );

    if(response.statusCode == 200)
    {
      final data = json.decode(response.body);
      APIRequestHelper.instance.accessToken = AccessToken(
        value: data['access_token'],
        type:  data['token_type'],
        expiresIn: data['expires_in'],
        refreshValue: data['refresh_token']
      );
    }

    print("Response: " + response.statusCode.toString());
  }
```

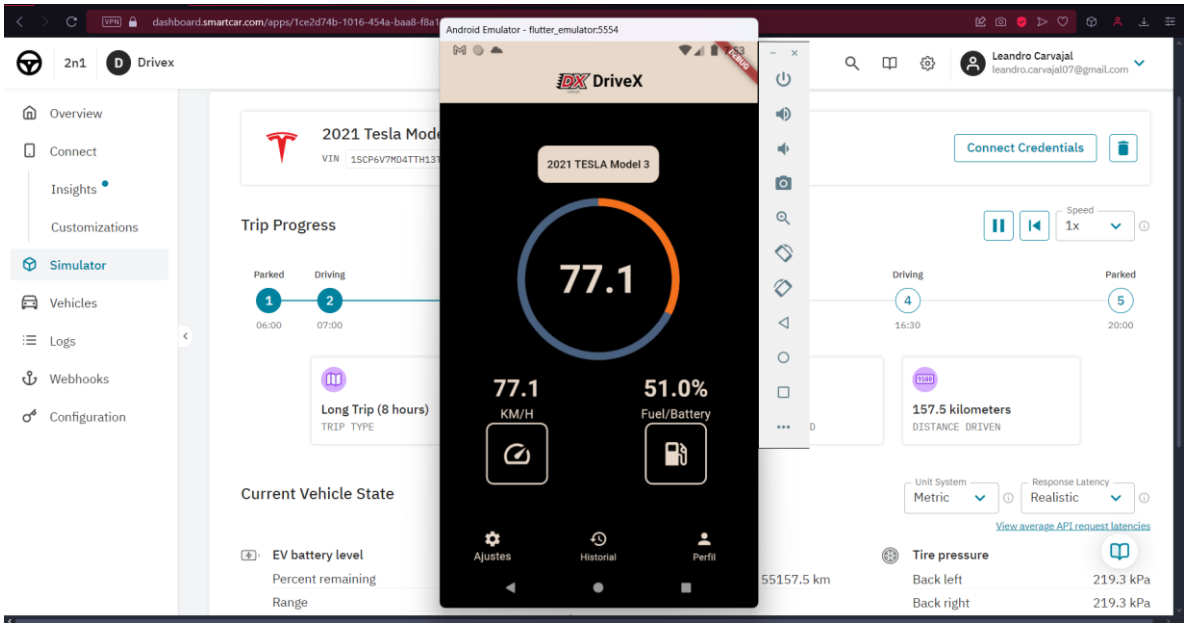
```
  static Future<void> loadUser() async
  {
    if(!instance.isConnected())
    {
      return;
    }

    Client client = Client();

    final Response response = await
client.get(Uri.parse("https://api.smartcar.com/v2.0/user"),
```

```
      headers: {
        'Authorization': 'Bearer
${APIRequestHelper.instance.accessToken!.value}'
      }
    );

    if(response.statusCode == 200)
    {
      var data = json.decode(response.body);
      User user = User(id: data['id']);
      List<User> users = await APIDatabaseHelper.getUsers();
      bool newUser = true;
      for(int i = 0; i < users.length; i++)
      {
        if(users[i].id == user.id)
        {
          newUser = false;
          break;
        }
      }
      if(newUser)
      {
        APIDatabaseHelper.insertUser(user);
      }
      APIRequestHelper.instance.user = user;
    }

    print("Respuesta user: " + response.statusCode.toString());
    return null;
  }
```

3. Prueba de obtención de información del vehículo

3.1 Tras realizar la conexión en menos de un minuto la aplicación obtiene la información relevante para la pantalla principal.

- Resultado esperado: Obtención de la información correspondiente al vehículo (Marca, modelo, año, odómetro y nivel del tanque).
- Resultado obtenido:



- Código implementado para el requerimiento:

```
static Future<Vehicle?> getVehicle(String id) async
  {
    if(!instance.isConnected())
    {
      return null;
    }

    Client client = Client();

    final Response response = await
client.get(Uri.parse("https://api.smartcar.com/v2.0/vehicles/$id"),
      headers: {
        'Authorization': 'Bearer
${APIRequestHelper.instance.accessToken!.value}'
      }
    );
```

```
    if(response.statusCode == 200)
    {
      var data = json.decode(response.body);
      return Vehicle(userID: APIRequestHelper.instance.user!.id, id:
data['id'], make: data['make'], model: data['model'], year: data['year']);
    }

    print("Respuesta vehiculo $id: " + response.statusCode.toString());
    return null;
  }
```

```
  static Future<void> loadVehicles() async
  {
    if(!instance.isConnected())
    {
      return;
    }

    Client client = Client();

    final Response response = await
client.get(Uri.parse("https://api.smartcar.com/v2.0/vehicles"),
      headers: {
        'Authorization': 'Bearer
${APIRequestHelper.instance.accessToken!.value}'
      }
    );

    if(response.statusCode == 200)
    {
      var data = json.decode(response.body);
      print(data['vehicles'].length);
      for(int i = 0; i < data['vehicles'].length; i++)
      {
        Vehicle? vehicle;
        do
        {
          vehicle = await getVehicle(data['vehicles'][0]);
          if(vehicle != null)
          {
            APIRequestHelper.instance.vehicles.add(vehicle);
            print(vehicle.id);
          }
```

```
        }while(vehicle == null);
      }

      List<Vehicle> vehicles = await APIDatabaseHelper.getVehicles();

      for(int i = 0; i < vehicles.length; i++)
      {
        await APIDatabaseHelper.insertVehicle(vehicles[i]);
      }
      print(data.toString());
    }
  }
```

```
  static Future<double?> getFuelRemaining(Vehicle vehicle) async
  {
    if(!instance.isConnected())
    {
      return null;
    }

    Client client = Client();

    final Response response = await
client.get(Uri.parse("https://api.smartcar.com/v2.0/vehicles/${vehicle.id}/f
uel" ),
      headers: {
        'Authorization': 'Bearer
${APIRequestHelper.instance.accessToken!.value}'
      }
    );

    if(response.statusCode == 200)
    {
      var data = json.decode(response.body);
      return data['percentRemaining'] * 100;
    };

    print("Respuesta combustible" + response.statusCode.toString());
    return null;
  }
```

```
  static Future<double?> getEVRemaing(Vehicle vehicle) async
  {
    if(!instance.isConnected())
```

```
    {
      return null;
    }

    Client client = Client();

    final Response response = await
client.get(Uri.parse("https://api.smartcar.com/v2.0/vehicles/${vehicle.id}/b
attery" ),
      headers: {
        'Authorization': 'Bearer
${APIRequestHelper.instance.accessToken!.value}'
      }
    );

    if(response.statusCode == 200)
    {
      var data = json.decode(response.body);
      switch(data['percentRemaining'])
      {
        case int i: return i.toDouble() * 100;
        case double d: return d * 100;
      }
    };

    print("Respuesta bateria" + response.statusCode.toString());
    return null;
  }
```

```
  static Future<double?> getOdometer(Vehicle vehicle) async
  {
    if(!instance.isConnected())
    {
      return 0;
    }

    Client client = Client();

    final Response response = await
client.get(Uri.parse("https://api.smartcar.com/v2.0/vehicles/${vehicle.id}/o
dometer"),
      headers: {
        'Authorization': 'Bearer
${APIRequestHelper.instance.accessToken!.value}'
```

```
    }
  );

  print("Respuesta odometro: ${response.statusCode}");
  if(response.statusCode == 200)
  {
    var data = json.decode(response.body);
    var value = data['distance'];

    switch(value)
    {
      case int i: return i.toDouble();
      case double d: return d;
      case String s: return double.parse(s);
    }
  }
```