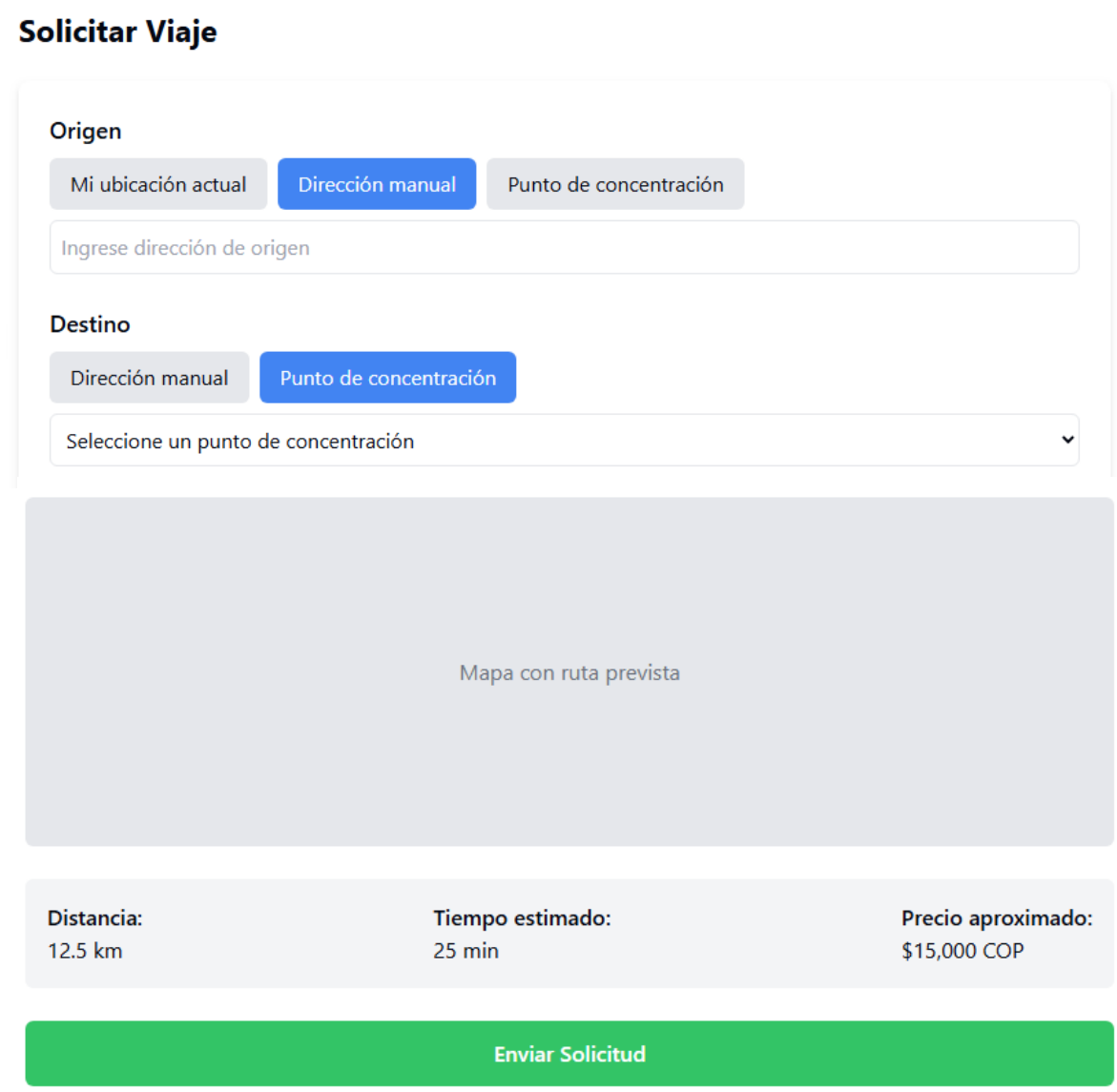
# FRONT-END

## HU007-Solicitar-viaje (Vista pasajeros)




### US-P01: Geolocalización Automática

La aplicación web debe detectar la ubicación actual del pasajero (HTML5 Geolocation + reverse-geocoding gratuito). De manera que si el pasajero está en un punto de alta concentración o punto particular sea detectado automáticamente *(esto comprobación debería realizarse desde el backend)*.

**Criterios de Aceptación**

- Usa la API de Geolocation del navegador
- Si el pasajero en **radio <100 m** de un **"punto de alta concentración"**, se marca automáticamente
- Si no, se le invita al pasajero a confirmar o teclear dirección

### US-P02: Selección de Destino

El pasajero puede seleccionar destino ya sea un punto de alta concentración (lista desplegable) o escribir dirección libre

**Criterios de Aceptación**

Lista con funcionalidad de búsqueda con los puntos de alta concentración de personas definidos.

Validación de mínimo un punto de alta concentración en origen o destino

## HU008-Activar-servicios (Vista conductores)

**Objetivo:** La idea es saber que conductores estan activos en un momento dado y dispónibles para realizar viajes (Ya que tenemos una lista de conductores, pero puede que muchos de estos conductores no esten activos en la plataforma, por lo que, es de vital importancia saber que conductores están activos y disponibles para tenerlos en encuenta en el empajeramiento de viajes)

**Implementación:**

- **Base de datos:**

Se ha creado una nueva tabla con las siguientes columnas:

```
TABLE public.conductores_activos_disponibles (
    conductor_id    INT    NOT    NULL    PRIMARY    KEY    REFERENCES
public.conductores(id) ON DELETE CASCADE,
    ubicacion_actual_lat DECIMAL(9,6) NOT NULL,
    ubicacion_actual_lon DECIMAL(9,6) NOT NULL,
    pmcp_cercano_id   INT   REFERENCES   public.puntos_concentracion(id)
DEFAULT NULL
    ultima_actualizacion_ubicacion TIMESTAMP WITHOUT TIME ZONE NOT NULL
DEFAULT CURRENT_TIMESTAMP
    sesion_expira_en TIMESTAMP WITHOUT TIME ZONE NOT NULL,
    estado_disponibilidad_viaje    VARCHAR(30)    NOT    NULL    DEFAULT
'disponible'   CHECK   (estado_disponibilidad_viaje   IN   ('disponible',
'ofrecido_viaje', 'en_viaje_asignado')),
    created_at TIMESTAMP WITHOUT TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP WITHOUT TIME ZONE DEFAULT CURRENT_TIMESTAMP
);
```

- **En el frontend:**

Cuando un conductor accede a la pagina de “Buscar viaje” y dicho conductor no se encuentra en la tabla “conductores_activos_disponibles” se debe mostrar a pantalla completa el siguiente mensaje “¿Estas interesado en ofrecer servicios de transporte compartido en estos momentos?”

“Activa tus servicios, botón: Haz clic aquí”

- **En el backend:**

Cuando el conductor hace clic en el botón “Haz clic aquí” el Backend hace un INSERT INTO conductores_activos_disponibles, y agrega el conductor, de esta manera se sabe que esta activo.

Se hará uso de dos implementaciones importantes:

- **Desactivación** (Automática/Cron Job): Un proceso backend ejecuta periódicamente: DELETE FROM conductores_activos_disponibles WHERE sesion_expira_en < NOW();. (Esto es para evitar que el estado de los conductores quede activo indefinidamente, la idea es que después de 1 hora, se elimine el estado de disponibilidad de los conductores (es decir, eliminado de la tabla “conductores_activos_disponibles” dicho registro de conductor.
- **Heartbeat**: Cada ping actualiza ubicacion_actual_lat/lon y ultima_actualizacion_ubicacion = NOW().

# BACK-END

**BK002-03- Optimización de Rutas y Cálculo de Métricas para Posibles Viajes**

**Trabajar en “bk002_03_optimizeTrips.js”**

**Objetivo:** Para cada "posible viaje" *(combinación de pasajeros - Ouput de la historia BK002-02)*. Se necesita determinar la secuencia óptima de paradas (recogidas/entregas), calcular la distancia total, el tiempo estimado y la ganancia estimada. La ruta debe ser la más corta posible minimizando desvíos.

**Tareas Técnicas:**

Para cada combinación de posibles_viajes_combinaciones:

1. Obtener las coordenadas del PMCP involucrado desde la tabla puntos_concentracion usando pmcp_id.
2. Construir la lista de waypoints:
    - Si pmcp_es_origen_del_grupo: El PMCP es el punto de inicio. Los siguientes waypoints son los destinos particulares de cada pasajero en la combinación.
    - Si pmcp_es_origen_del_grupo es false: Los waypoints iniciales son los orígenes particulares de cada pasajero. El PMCP es el punto final común.
3. **Optimización de Secuencia de Paradas (Problema del Viajante - TSP):**
    - Si pmcp_es_origen_del_grupo: El PMCP es fijo como inicio. Se necesita encontrar el orden óptimo para visitar los N destinos.

- Si pmcp_es_origen_del_grupo es false: Se necesita encontrar el orden óptimo para visitar los N orígenes. El PMCP es fijo como final.
- Para N entre (la capacidad mínima y máxima de los vehiculo de conductores disponibles), se pueden explorar algoritmos exactos (si se usa una matriz de distancias precalculada entre todos los puntos) o heurísticas eficientes (Nearest Neighbor, algoritmos genéticos simples).
- Utilizar un servicio de ruteo (ej. OSRM auto-hosteado, o API pública de OpenRouteService/GraphHopper con límites de uso gratuito) para obtener la matriz de distancias/tiempos entre todos los puntos relevantes (PMCP y orígenes/destinos particulares de la combinación).

4. Una vez obtenida la secuencia óptima de paradas, usar el servicio de ruteo nuevamente para obtener la ruta completa (GeoJSON), la distancia total y el tiempo estimado para esa secuencia.
5. Calcular la ganancia estimada: Ganancia = (Suma de tarifas de pasajeros en la combinación) - (Costo operativo del viaje). El costo operativo podría ser un % de la tarifa o basado en distancia/tiempo. Se necesita definir una política de tarifas por pasajero (por distancia).

**Flujo de trabajo**

1. Lee combinaciones_viaje_propuestas con estado 'optimizacion_pendiente'.
2. Realiza la optimización de ruta y cálculo de métricas.
3. Si la combinación es viable y cumple criterios (rentabilidad, etc.):
4. Crea un nuevo registro en la tabla viajes (con estado = 'disponible', conductor_id = NULL).
5. Crea los registros correspondientes en viaje_pasajeros (usando solicitud_viaje_id de solicitudes_en_combinacion_propuesta).
6. Actualiza el estado de las solicitudes_viaje originales a 'ofertado' (o similar, indicando que ahora forman parte de un viaje potencial).
7. Actualiza combinaciones_viaje_propuestas.estado_procesamiento a 'optimizada_oferta_creada'.
8. Si no es viable, actualiza combinaciones_viaje_propuestas.estado_procesamiento a 'descartada_...'

**Input (Output BK002-02)**

```
{
  "posibles_viajes_combinaciones": [
    {
      "pmcp_id": 1,
      "pmcp_es_origen_del_grupo": true,
      "combinaciones": [
        {
          "pasajeros_participantes": [
            // Combinación de 3 pasajeros
```

```
            {
              "solicitud_id": 1,
              "pasajero_id": 101,
              "origen_lat": 4.5,
              "origen_lon": -74.1,
              "destino_lat": 4.8,
              "destino_lon": -74.3
            },
            {
              "solicitud_id": 2,
              "pasajero_id": 102,
              "origen_lat": 4.5,
              "origen_lon": -74.1,
              "destino_lat": 4.9,
              "destino_lon": -74.4
            },
            {
              "solicitud_id": 5,
              "pasajero_id": 105,
              "origen_lat": 4.5,
              "origen_lon": -74.1,
              "destino_lat": 4.7,
              "destino_lon": -74.2
            }
          ],
          "capacidad_utilizada": 3
        },
        {
          "pasajeros_participantes": [
            // Otra combinación de 3
            {
              "solicitud_id": 1,
              "pasajero_id": 101,
              "origen_lat": 4.5,
              "origen_lon": -74.1,
              "destino_lat": 4.8,
              "destino_lon": -74.3
            },
            {
              "solicitud_id": 2,
              "pasajero_id": 102,
              "origen_lat": 4.5,
              "origen_lon": -74.1,
              "destino_lat": 4.9,
              "destino_lon": -74.4
            },
            {
              "solicitud_id": 6,
              "pasajero_id": 106,
              "origen_lat": 4.5,
              "origen_lon": -74.1,
              "destino_lat": 4.6,
              "destino_lon": -74.0
            }
          ],
```

```
          "capacidad_utilizada": 3
        },
        {
          "pasajeros_participantes": [
            // Combinación de 4 pasajeros
            {
              "solicitud_id": 1,
              "pasajero_id": 101,
              "origen_lat": 4.5,
              "origen_lon": -74.1,
              "destino_lat": 4.8,
              "destino_lon": -74.3
            },
            {
              "solicitud_id": 2,
              "pasajero_id": 102,
              "origen_lat": 4.5,
              "origen_lon": -74.1,
              "destino_lat": 4.9,
              "destino_lon": -74.4
            },
            {
              "solicitud_id": 5,
              "pasajero_id": 105,
              "origen_lat": 4.5,
              "origen_lon": -74.1,
              "destino_lat": 4.7,
              "destino_lon": -74.2
            },
            {
              "solicitud_id": 6,
              "pasajero_id": 106,
              "origen_lat": 4.5,
              "origen_lon": -74.1,
              "destino_lat": 4.6,
              "destino_lon": -74.0
            }
          ],
          "capacidad_utilizada": 4
        }
        // ... más combinaciones
      ]
    }
    // ... combinaciones para otros grupos de PMCP
  ]
}
```

**Output**

```
{
  "viajes_potenciales_optimizados": [
    {
      "pmcp_id": 1,
      "pmcp_es_origen_del_grupo": true,
      "pasajeros_participantes": [
```

```
        // IDs de las solicitudes originales o pasajeros_id
        {
          "solicitud_id": 1,
          "pasajero_id": 101
        },
        {
          "solicitud_id": 2,
          "pasajero_id": 102
        },
        {
          "solicitud_id": 5,
          "pasajero_id": 105
        }
      ],
      "capacidad_utilizada": 3,
      "orden_paradas_optimizado": [
        // Objetos con tipo (recogida/entrega) y coords, o IDs
        // Ejemplo si PMCP es origen:
        {
          "tipo": "pmcp_origen",
          "lat": 4.5,
          "lon": -74.1,
          "pmcp_id": 1
        },
        {
          "tipo": "entrega",
          "pasajero_id": 105,
          "lat": 4.7,
          "lon": -74.2
        }, // Supongamos D3 es el más cercano
        {
          "tipo": "entrega",
          "pasajero_id": 101,
          "lat": 4.8,
          "lon": -74.3
        }, // Luego D1
        {
          "tipo": "entrega",
          "pasajero_id": 102,
          "lat": 4.9,
          "lon": -74.4
        }  // Finalmente D2
      ],
      "ruta_geojson": {
        "type": "LineString",
        "coordinates": [
          [-74.1, 4.5],
          [-74.2, 4.7]
          // ... más coordenadas
        ]
      },
      "distancia_km": 12.5,
      "tiempo_estimado_min": 25,
      "ganancia_estimada_cop":  15000  //  (Ej:  3  pasajeros  *  7000
tarifa_prom - 6000 costo_op)
```

```
    }
    // ... más viajes potenciales optimizados
  ]
}
```