EVTC sur LMFS

1. Appel Middle Office API en utilisant HTTP request python

Dans le module middle_office_connector, on a un modèle “middle.office”. Et après on a des fonction pour avoir les URL de chaque type d’appel

par exemple :

- avoir l’url https://movtc-billing-calculation-bp3cgjjccq-ew.a.run.app/trip-mapping/abort-trip
- avec la fonction :

```
def get_uri_cancel_trip(self):
    return self.search([
        ('type', '=', 'cancel_trip')
    ], limit=1)
```

Après pour pouvoir construire une requête HTTP de tous type (GET, POST, ..) avec l’url, on utilise :

```
def send_requests_to_mo(self, payload=None, params=None, request="POST", paramstypes=None, name=''):
    if not payload:
        payload = json.dumps({})
    headers = self.get_header_application()
    uri_with_params = self.get_uri_with_params(params=params,
                                               url=self.get_full_url().strip(),
                                               paramstypes=paramstypes)
    if request == "GET":
        response = requests.request(request, uri_with_params, headers=headers)
    else:
        response = requests.request(request, uri_with_params, headers=headers, data=payload)
    self.set_logs(body=payload, url=uri_with_params, type=request, response=response, headers=headers, name=name)
    return response
```

- payload : body request
- params : paramètre dans l’url
- request : type POST ou GET
- self.get_header_application () : pour avoir l’header du requete

Après que la requête est envoyée, on enregistre tous les appels en créant l'objet “ request.lmfs.log” , avec la fonction :

```python
def set_logs(self, headers, url, name, type, response, body):
    name = "%s %s %s" % (name, type, url)
    values = {
        "types": type.upper(),
        "headers": headers,
        "response_header": response.headers,
        "response_content": response.text,
        "request_body": body,
        "url": url,
        "name": name.strip(),
        "code": response.status_code,
        "content": response.content,
        "reason": response.reason,
        "encodings": response.encoding,
        "date": datetime.now().strftime("%d-%m-%Y %H:%M"),
        "response": response,
    }
    self.env['request.lmfs.log'].create(values)
```

2. Affectation depuis Middle Office

Dans project/evtc-addons/evtc_lmfs/wizards/auto_planning_wizard.py, on a le fonction “default_get”.

```python
@api.model
def default_get(self, fields):
    planning = super(AutoPlanningWizard, self).default_get(fields)
    record_id = self.env['crm.lead'].sudo().browse(self._context.get('active_ids'))
    role_mo_service = self.get_role_by_immatricule({
        "lat": record_id.pick_up_lat,
        "lon": record_id.pick_up_long
    })
    planning_array = planning['planning_ids']
    for array in planning_array:
        if array[2]:
            role_id = array[2].get('role_id', False)
            mo_state = get_value_by_search(role_mo_service, 'vehicle_id', role_id, 'status')
            array[2]['estimated_distance'] = get_value_by_search(role_mo_service, 'vehicle_id', role_id, 'estimate
            array[2]['estimated_time_arrival'] = float_to_time(get_value_by_search(role_mo_service, 'vehicle_id',
            array[2]['mo_state'] = mo_state
    planning['planning_ids'] = sorted(planning_array, key=lambda x: (
        x[2].get('estimated_time_arrival', "--:--:--") == "--:--:--",
        x[2].get('estimated_time_arrival', "--:--:--")
    ))
    return planning
```

- self.get_role_by_immatricule({lat, long}) : avoir les les recommandation via MO, avec les distance estimatif et temp d’arriver estimatif avant d’arriver au client

```python
def get_role_by_immatricule(self, values):
    role = []
    vehicle_mo = self.get_mo_vehicle_recommandation(values)
    if vehicle_mo.text == '[]' : return role
    data_array = json.loads(vehicle_mo.text)
    for vehicle in data_array:
        if vehicle['status'] not in ['AVAILABLE','BUSY']:
            continue
        vehicle_id = self.env['planning.role'].sudo().search([('vehicle_id.license_plate','=',vehicle.get('immatri
        dict_temp = {
            'vehicle_id': vehicle_id.id,
            'estimateDistance': vehicle['estimateDistance'],
            'estimateTimeofArrival': vehicle['estimateTimeofArrival'],
            'status': vehicle['status']
        }
        if vehicle_id: role.append(dict_temp)
    return role
```

Après l'appel self.get_mo_vehicle_recommandation(values) : on a ce reponse

[{ "status": "AVAILABLE", "position": { "latitude": -18.871653288264834, "longitude": 47.50999242067337 }, "estimateDistance": 0.05296472798115732, "estimateTimeofArrival": 9.533651036608317 }, { "immatriculation": "60429WWT", "status": "AVAILABLE", "position": { "latitude": -18.868454, "longitude": 47.5139847 }, "estimateDistance": 0.548262823162993, "estimateTimeofArrival": 98.68730816933875 }, { "immatriculation": "50239WWT", "status": "AVAILABLE", "position": { "latitude": -18.8738308, "longitude": 47.5179024 }, "estimateDistance": 0.908734539460251, "estimateTimeofArrival": 163.57221710284517 }, { "immatriculation": "HONDA45940WWT", "status": "AVAILABLE", "position": { "latitude": -18.8737793, "longitude": 47.5179253 }, "estimateDistance": 0.9092444760808847, "estimateTimeofArrival": 163.66400569455922 }, { "immatriculation": "48135WWT", "status": "AVAILABLE", "position": { "latitude": -18.8739319, "longitude": 47.5179024 }]

```python
def get_value_by_search(list_of_dicts, search_key, search_value, target_
    for dictionary in list_of_dicts:
        if dictionary.get(search_key) == search_value:
            return dictionary.get(target_key)
    return None
```

cette fonction a pour but de chercher des valeurs en entrant :

- le dictionnaire
- cle pour être sélectionné
- la valeur du cle
- le cle cible dans le dictionnaire

Cette fonction est utilisée pour chercher des valeurs spécifiques au moment de la recommandation.

3. Création REST API sur ODOO

Apres installation REST API (module tier), on peut créer une "endpoint" pour être consommé :

- Creer un datamodel

```python
from odoo.addons.datamodel import fields
from odoo.addons.datamodel.core import Datamodel


class TripInfos(Datamodel):
    _name = 'trip.infos'

    siid = fields.String(allow_none=True)
    opportunity = fields.Dict(allow_none=True)
    orders = fields.Dict(allow_none=True)
    api_key = fields.String(allow_none=True, required=True)
```

ces champs sont responsable des cles dans le json envoyer par middle office

- Apres on crée un service,

```python
class OpportunityCallback(Component):
    _inherit = 'base.rest.service'
    _name = 'trip.update'
    _usage = 'trip'
    _collection = 'trip.callbacks'
```

dans ce service, on a une méthode avec la décoration

```python
@restapi.method([(['/update'], 'POST')], auth='lmfs_api_key', type='json',
website=False,
input_param=Datamodel('trip.infos'))
```

C’est le endpoint, comme dans les contrôleurs odoo sauf qu’ici on l’appel pas sur web mais en HTTP request

La fonction en dessous du décorateur c’est la function qui est utilisé pour mettre à jour les champs du CRM depuis un consommateur.

```python
def update_trip_infos(self, trip_infos):
    opportunity_id = request.env['crm.lead'].sudo().search([
        ('siid', '=', trip_infos.siid)
    ])
    values = {
        "types": "POST",
        "headers": "",
        "response_header": "",
        "response_content": "",
        "request_body": trip_infos,
        "url": "/callbacks/trip/update",
        "name": "update",
        "code": "",
        "content": "content",
        "reason": "",
        "encodings": "",
        "date": "",
        "response": "",
    }
    test = request.env['request.lmfs.log'].create(values)
    if opportunity_id.role_id and opportunity_id.role_id.vehicle_id.geolocalization_type == 'geotab':
        return {
            'status_code': 200,
            'message': _('Vehicle localisation: GEOTAB')
        }
    if not opportunity_id:
        raise BadRequest(_('Ivalid  siid'))
    orders = opportunity_id.order_ids.filtered(lambda s: s.state == 'draft')
    order_id = orders[-1] if len(orders) > 1 else orders
    if not orders and trip_infos.orders:
        raise BadRequest(_('No order found for this trip'))
    request_data = ''
    if trip_infos.opportunity:
        request_data += self._translate_error_callback_message(
            opportunity_id.set_opportunity_request(trip_infos.opportunity)
        )
    if not request_data and trip_infos.orders:
```

4. Réaffectation d’une course :

Si un véhicule déjà affecté à une course s’est changé en statue indisponible, il y a un bouton assigner qui apparaît dans le statut course en cours.

```xml
<!-- Reassigning view -->
<record id="crm_lead_view_kanban_reassigning_inherit" model="ir.ui.view">
  <field name="name">crm.lead.view.kanban.reassigning.inherit</field>
  <field name="model">crm.lead</field>
  <field name="inherit_id" ref="crm.crm_case_kanban_view_leads" />
  <field name="arch" type="xml">
    <xpath expr="//button[@name='action_cancel_stage']" position="after">
      <field name="role_has_down"  invisible='1'/>
      <button name="%(etech_auto_planning.action_auto_planning)d" type="action"
        class="btn btn-primary"
        attrs="{'invisible':['|',('state_value','not in',['crm.stage_lead2','crm.stage_lead3']),('role_has_down','
        Reassigning
      </button>
    </xpath>
  </field>
</record>
```

5. POS controller :

Pour accéder directement dans le POS d’un véhicule, on a créé une controlleur dans project/evtc-addons/evtc_lmfs/controllers/pos_controller.py.

lien : baseURL/pos/user?immatriculeID=9999WWT

- Cherche les POS lie au matricule :

```
role_by_immatricule =
request.env['planning.role'].sudo().search([('vehicle_id.license_plate'
,'=',immatriculeID)])
```

- et apres on recupere le POS_CONF par pos
- on ouvre le POS par rapport à son statut

```
if not current_pos:
    return Response(template='website.page_404', status=404)
current_pos = current_pos[0]
if current_pos.current_session_state == 'opened':
    val = current_pos.open_ui()
    return werkzeug.utils.redirect(val.get('url',''))
if current_pos.current_session_state == 'opening_control':
    current_session = current_pos.current_session_id
    val = current_session.open_frontend_cb()
    return werkzeug.utils.redirect(val.get('url',''))
if current_pos.current_session_state == 'closing_control':
    current_session = current_pos.current_session_id
    val = current_session.open_frontend_cb()
    return werkzeug.utils.redirect(val.get('url',''))
if not current_pos.current_session_id and  not current_pos.pos_session_username:
    val = current_pos.open_session_cb()
    return werkzeug.utils.redirect(val.get('url',''))
```

- Modification des boutons start, pause, et stop dans POS : si lmfs, on les caches
    - Création fonction pour avoir le type de géolocalisation dans le modèle pos.session:

```
class FleetVehicle(models.Model):
    _inherit = 'pos.session'

    def get_geolocalization_type(self, role_id):
        role_id = self.env['planning.role'].browse(role_id.get('id'))
        if role_id:
            vehicle_id = role_id.vehicle_id
            if vehicle_id.geolocalization_type:
                return vehicle_id.geolocalization_type
            else:
                return ""
```

    - dans project/evtc-addons/esanandro_geotab/static/src/js/pos_reserv_js.js, on intègre cette fonction

```
class GeotabPosOrder extends PosComponent {
    constructor() {
        super(...arguments);
        useListener('click', this.clickStart);
    }

    async get_roleID_geo_type(){
        await rpc.query({
            model:'pos.session',
            method: 'get_geolocalization_type',
            args: [[odoo.pos_session_id],{'id':this.env.pos.config.role_id[0]}]
        },{
            timeout: 3000,
            shadow: true
        }).then(res=>{
            if(res == "lmfs"){
                $("#start-btn").remove()
                $("#stop-btn").remove()
                $("#pause-btn").remove()
            }
        },error=>{
            console.error(error)
            return ""
        })
    }
```

6. Controlleur trip-reservation (map client)

le controlleur se strouve dans : /project/evtc-addons/evtc_lmfs/controllers/trip_progress.py

url : baseURL/trip-reservation/<crm_id>

```
values = opportunity_id.prepare_dashboard_values()
```

c'est d'avoir toutes les données du CRM affiché sur le front client.

```
progress_values =
opportunity_id.with_user(SUPERUSER_ID).prepare_progress_values()
```

création d'un dictionnaire pour la vue progression sur le map client

```python
def prepare_progress_values(self):
    crm_stage = self.get_crm_stage()
    values = {}
    progressions = [
        ('50', '0', '0'),('75','0','0'),('100','0','0'),
        ('100', '25', '0'),('100', '50', '0'),('100', '100', '0'),
        ('100', '100', '100'),('0', '0', '0')
    ]
    stages = {
        crm_stage['new_request']: (_('Request being validated'), 1, 'step-one', progressions[0]),
        crm_stage['trip_confirmed']: (_('Your trip is confirm'), 2, 'step-two', progressions[1]),
        crm_stage['trip_affected']: (_('Your trip is affected'), 10, 'step-three', progressions[2]),
        crm_stage['driver_coming']: (_('Your driver is on his way'), 3, 'step-three', progressions[3]),
        crm_stage['end_pick_up_client']: (_('You have been picked up'), 3, 'step-three', progressions[4]),
        crm_stage['in_progress']: (_("trip in progress"), 4, 'step-four', progressions[5]),
        crm_stage['compta']: (_("trip finish"), 4, 'step-four', progressions[6]),
        crm_stage['terminated']: (self.stage_id.name, 4, 'step-four', progressions[6]),
        crm_stage['refused']: (_('The race is refused'), 3, 'step-three', progressions[7])
    }
    if self.stage_id in stages:
        state_label, step, class_step, progression = stages[self.stage_id]
        values.update({
            'state_label': state_label,
            'step': step,
            'class_step': class_step,
            'progression': progression
        })
    else:
        values.update({
            'state_label': 'Unknown step',
            'step': 1,
            'class_step': 'step-one',
            'progression': progressions[0]
        })
    return values
```

variable :

- progressions = [(50,0,0)] veut dire dans le front : 50/100 step 1 - 0/100 step 2 - 0/100 step 3
- stages = {
  crm_stage[‘new_request’] : clé du dictionnaire (status)
  (_(Reqesst …)) : text a afficher a chaque étape
  1 : numéro de l'étape
  ‘step-one’: nom de l'étape
  progression[0] : line de progression à être affichée
  }

sur le front, dans le template /project/evtc-addons/evtc_lmfs/views/progress_trip/index.xml

```
<div id="evtc_aside"
    t-att-data-id="crm_id"
    t-att-class="'sidebar col-sm-5 col-12 order-2 order-sm-1 pb16 pt16 '+ class_step">
    <t t-call="evtc_lmfs.trip_step_state_container" />
    <h2 class="taxi-heading-2 text-center">
        <t t-esc="state_label" />
    </h2>
    <t t-if="step in (1,2,3,4,10)" t-call="evtc_lmfs.trip_step_body" />
</div>
<div id="evtc_maps" class="ccol-12 col-sm-7 order-1 order-sm-2 map_evtc" style="position:
    <t t-if="step in (1,2,4,10)">
        <img class="detail_course_map"
            t-att-src="'/web/image/crm.lead/'+ str(crm_id) +'/maps_image'"
            alt="CRM maps image"
        />
    </t>
    <t t-elif="step == 3">
        <t t-call="evtc_lmfs.reservation_map_views"/>
    </t>
</div>
```

7. Synchronisation des véhicules vers MO
    A. creation véhicule

```
@api.model_create_multi
def create(self, values):
    to_found = isinstance(values, list) and values[0] or values
    vehicle_id = super(FleetVehicle, self).create(values)
    if any(element in to_found for element in self.get_vehicle_sync_fields()):
        vehicle_id.create_mo_vehicle()
    return vehicle_id
```

```
- if any(element in to_found for element in
  self.get_vehicle_sync_fields()):
```

- Avoir tous les champs requis pour l'envoie vers MO

```
def create_mo_vehicle(self):
    mo_data = self.get_mo_json_values()
    mo_default = self.env['middle.office'].search([('type', '=', 'create_vehicle')],limit=1)
    response = False
    if mo_default:
        response = mo_default.send_requests_to_mo(mo_data)
        if response.status_code != 200:
            raise ValidationError(_('requests failed, please see log for more details'))
    return response
```

- Get default mo API URL et après on appel send_rquest_to_mo() : fonction pour créer une requête HTTP sur python.

- Mise à jour statuts véhicule vers Middle Office:

```python
def update_state_vehicle_mo(self, license_plate, status):
    mo_default = self.env['middle.office'].search([('type', '=', 'update_state_vehicle')],limit=1)
    response = False
    mo_data = json.dumps({
        "immatriculation": license_plate,
        "status": status
    })
    if mo_default:
        response = mo_default.send_requests_to_mo(mo_data)
        if response.status_code != 200:
            raise ValidationError(_('requests failed, please see log for more details'))
    return response
```

le statut de chaque véhicule doit etre toujour sync vers MO, pour cela on a creer un champ qui contient le statut qui est universel avec ODOO et MO

```python
class FleetVehicleState(models.Model):
    _inherit = 'fleet.vehicle.state'

    is_broken_down = fields.Boolean("Indisponible")
    middle_office_value = fields.Selection([
        ('AVAILABLE','AVAILABLE'),
        ('UNAVAILABLE','UNAVAILABLE')
    ], string="Middle Office State", default="AVAILABLE")


    @api.onchange('is_broken_down')
    def _onchange_is_broken_down(self):
        if self.is_broken_down:
            self.write({
                'middle_office_value': 'UNAVAILABLE'
            })
        else:
            self.write({
                'middle_office_value': 'AVAILABLE'
            })
```

8. Synchronisation chauffeur vers MO :

A chaque fois qu'on crée ou modifie un contact et que le champ "est chauffeur " est coché, on doit faire appel deux API de MO (création de contact, création de véhicule pour ce chauffeur) .

- Fonction qui appel les deux API

```python
def sync_payload(self):
    payload = self.get_payload()
    self.sync_partner(
        self.convert_key(payload)
    )
    self.sync_vehicle_driver(
        payload['phone']
    )
```

- Fonction création contact :

```python
@api.model_create_multi
def create(self, vals_list):
    result = super(ResPartner, self).create(vals_list)
    if result.is_driver:
        result.sync_payload()
    return result
```

- Fonction update contact :

```python
def write(self, values):
    required_fields = ['name', 'image_1920', 'phone', 'is_driver']
    to_update = False
    for f in required_fields:
        if f in values:
            to_update = True
    super(ResPartner, self).write(values)
    if to_update:
        self.sync_payload()
```

9. Script position :

Pour remonter les position des chaque véhicules, on doit lancer un script à part ODOO pour l'insérer directement dans REDIS.

python script.py -hr localhost -p 6379 -u 'm.rakotoarisoa@etechconsulting-mg.com' -d 'etechconsulting' -pwm 'GeoTab#2021!eTech$ESA' -sg 'my1194.geotab.com' -mo 'https://modis.qualif.arkeup.com/position_management/vehicle-mapping/all-position' -key 'apikey C603LtrigedeprO1ux'

Pour le lancer, on doit appeler le script dans /script/script.py avec les arguments :

- -hr : server REDIS
- -p : post REDIS
- -u : login username geotab
- -d : database geotab
- -pwm : mot de passe geotab
- -sg : base URL geotab
- -mo : Lien MO pour récupérer tous les positions des vehicules
- -key : API KEY MO (clé d'authorisation)

Nom du branch pour MEP LMFS : MEP/LMFS

COMMIT qui ne sont pas dans le MEP/LMFS:

Traduction stage d2734e69a0b56a531436db6bbeced4f02a4b43ae

Correction css 3cb40a26a03a5a316e50540c2a125efd47ec09df

Correction responsive map client cee1a52340cab999c8104906ec3b0d04b73649d5

Ajouter synchro status vehicule da73acb08d446e6a3ab1f2cb77530555fcc68842

Afficher map en step 3 seulement db52e09bcfd40a39a6e48954db8623e6a72ad135

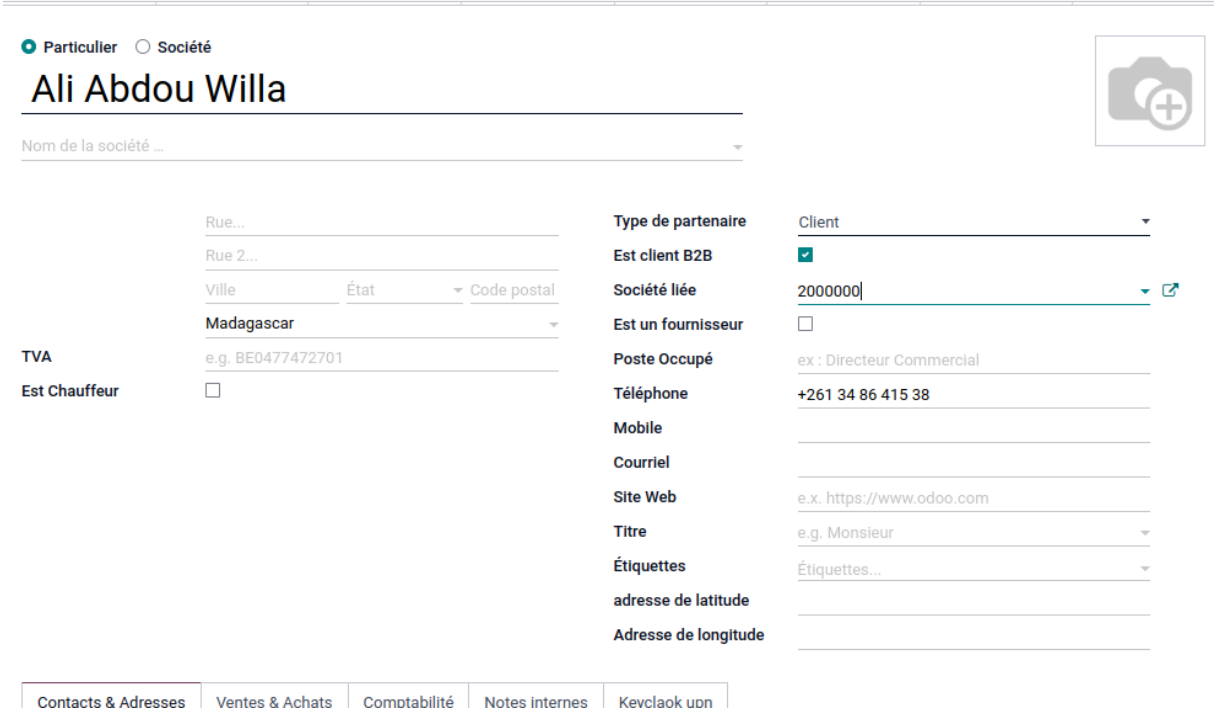
Fix afficher map step 3 f1ba8972e517507a3e8a03b6201c2c13f9bb364d

Reset map Responsive 5a43e48f82fa30d609ae80fe4e480daae246c32a

1. Permettre de savoir les clients btob

Module MAJ : etech_auto_planning (maj au démarrage requis)

Ajout nouvelle champs est client b2b et société liée dans le contact des utilisateurs

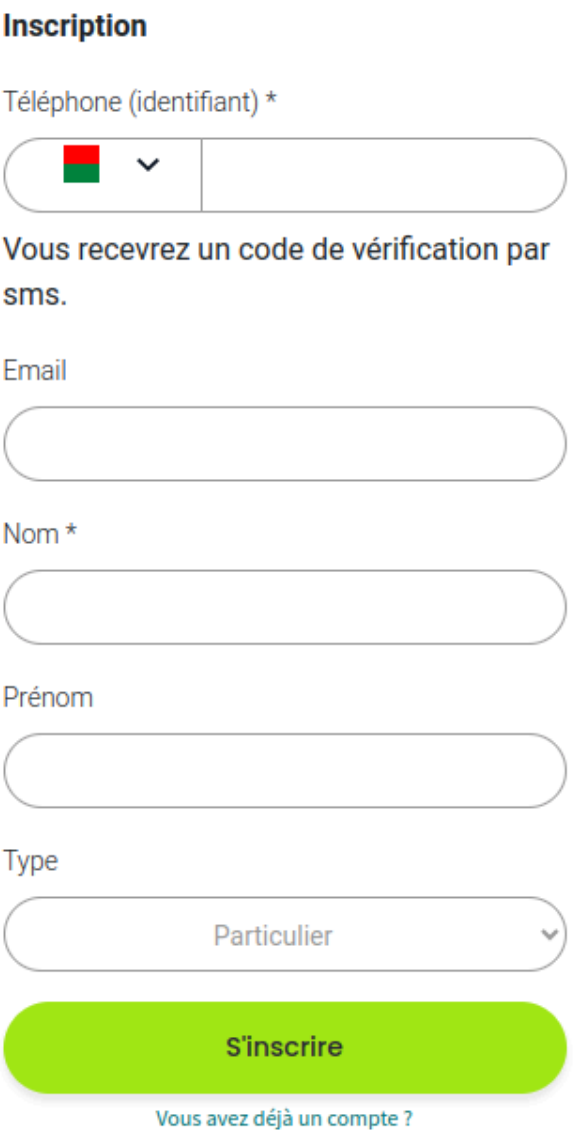


2. Permettre de rattacher au devis le contact de facturation dans le cas d'une réservation d'un client Btob lors de la validation

3. Permettre de télécharger les relevés de la course sous format pdf et excel

Ajouter une fonctionnalité au front pour les clients de télécharger les listes de course en pdf ou en excel

4. Permettre de faire une réservation pour ma société



Type si c'est particulier ou Société, dans le cas c'est sociétés on choisit une société pour être lié dans le système

5. Permettre de synchroniser les utilisateurs Odoo vers Keycloak dans le backoffice

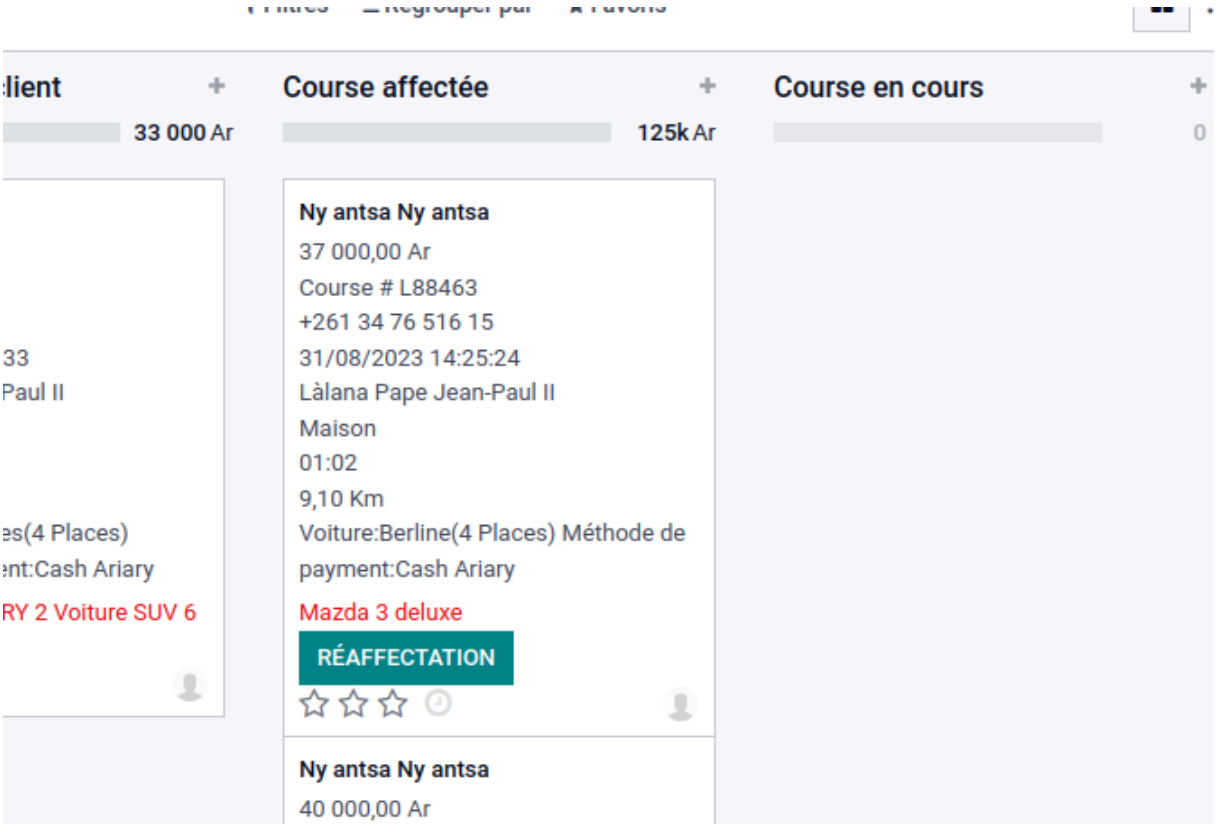
Après la modification du mot de passe de chaque utilisateur, on envoie tous les donnees de l'utilisateur vers keycloack

Voir doc keycloack pour connaître les endpoints de keycloack.

6. Permettre d'avoir une étape de réaffectation s'il y a une erreur d'affectation

Ajout champs dans le statut des véhicules pour savoir quel est l'etape qui n'est pas disponible

si la voiture affecte change en statue pas dispo, dans CRM, le bouton réaffectation sera visible:



Si on clique sur le bouton, le système ouvre une modal pour faire une affectation, la seule différence avec l'affectation c'est seulement le modèle de SMS et l'email à envoyer vers les clients.

7. Permettre d'ajouter la référence de la course sur le template sms lors des étapes : Nouvelle course, course confirmée, course affectée, course refusée, course annulée

8. Permettre d'avoir le dernier chauffeur du véhicule sélectionné lors de l'affectation

Branche : Feature-ESANTMA-344

Ajoute une popup au moment du validation de l’affectation