MINISTÈRE
DES ARMÉES
ET DES ANCIENS
COMBATTANTS

Liberté
Égalité
Fraternité

RAPPORT DE STAGE BTS

Services Informatiques aux Organisations

2nd année – 2025-2026

ROCHER Maxime

12 janvier 2026 au 15 juin 2026

Entreprise d'accueil : Cabinet de la Ministre des Armées et des Anciens Combattants

Établissement : Lycée La Colinière
Enseignant référant : S. DION

Sommaire

1. Remerciements

Je remercie l'ensemble du Cabinet de la Ministre des Armées et des Anciens Combattants pour m'avoir accueilli et intégré dans de très bonnes conditions.

Je tiens à adresser un remerciement particulier à ma Commandant pour la confiance qu'elle m'a accordée durant ce stage ainsi que pour son encadrement et ses conseils.

Je remercie également l'équipe de développement pour sa disponibilité, sa pédagogie et sa bienveillance. Les échanges quotidiens, les explications techniques et l'aide apportée lorsque je rencontrais des difficultés m'ont permis d'approfondir mes compétences en développement et de progresser sur le plan technique. Leur soutien m'a aidé à gagner en autonomie et en rigueur.

Je souhaite aussi remercier l'ensemble des personnels civils et militaires avec qui j'ai eu l'occasion d'échanger. Leur gentillesse, leur bienveillance et la qualité des discussions ont largement contribué à rendre cette expérience agréable et enrichissante.

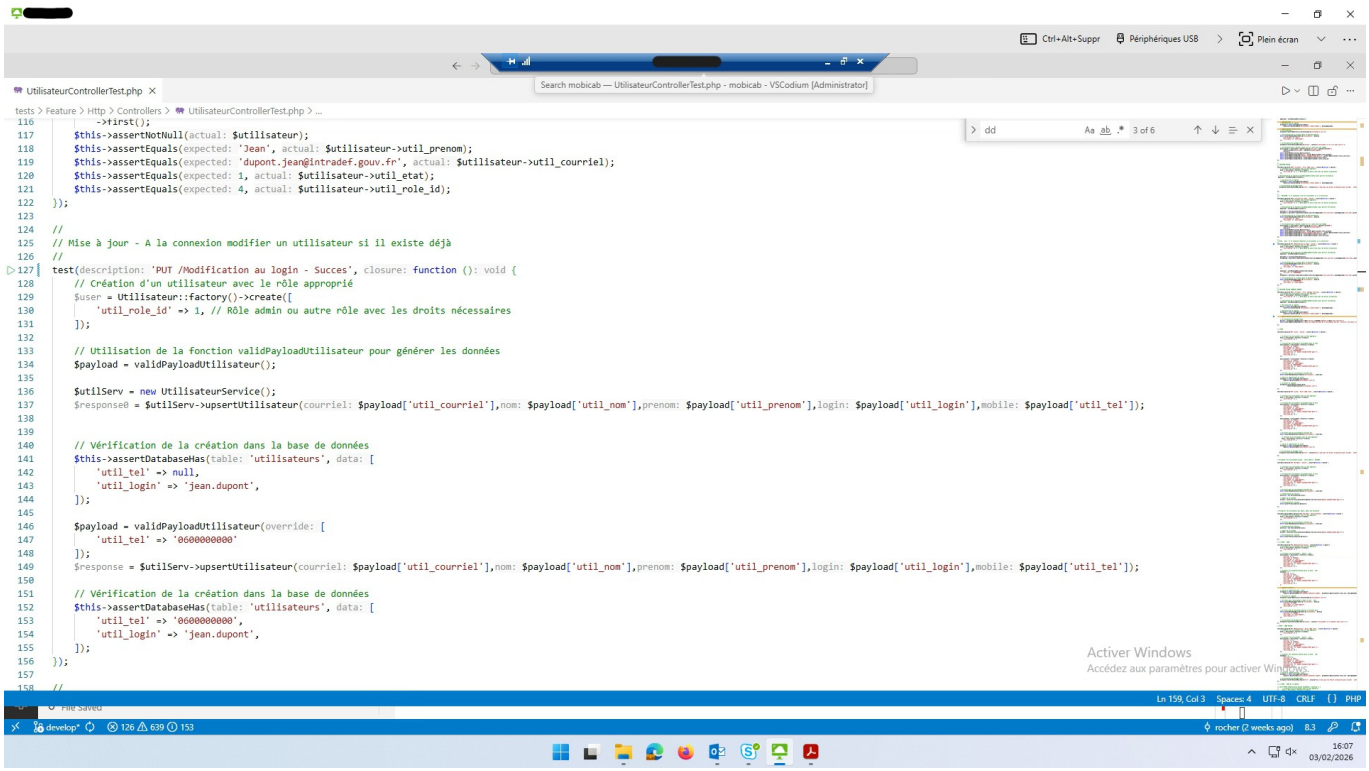
Je remercie enfin mon enseignant référent ainsi que l'équipe pédagogique pour leur accompagnement tout au long de l'année.

2. Environnement de travail et de l'entreprise

Accueilli à mon arrivée par le personnel, une visite des locaux de l'établissement m'a été faite. Cette première étape a permis de rencontrer les différentes personnes, civiles et militaires, avec lesquelles je travaillerai. Une sensibilisation aux règles de conduite et de confidentialité m'a également été donnée, précisant ce qui est autorisé ou non.

Un point qui m'a particulièrement marqué concerne l'usage des clés USB : seules celles fournies par une personne habilitée sont autorisées. Chaque clé USB doit être analysée sur une station blanche, reliée à un VLAN dédié et sécurisé, équipée d'un antivirus. Bien que connectée au réseau, cette station est isolée des autres environnements afin d'empêcher toute propagation de virus ou de logiciels malveillants vers le système d'information.

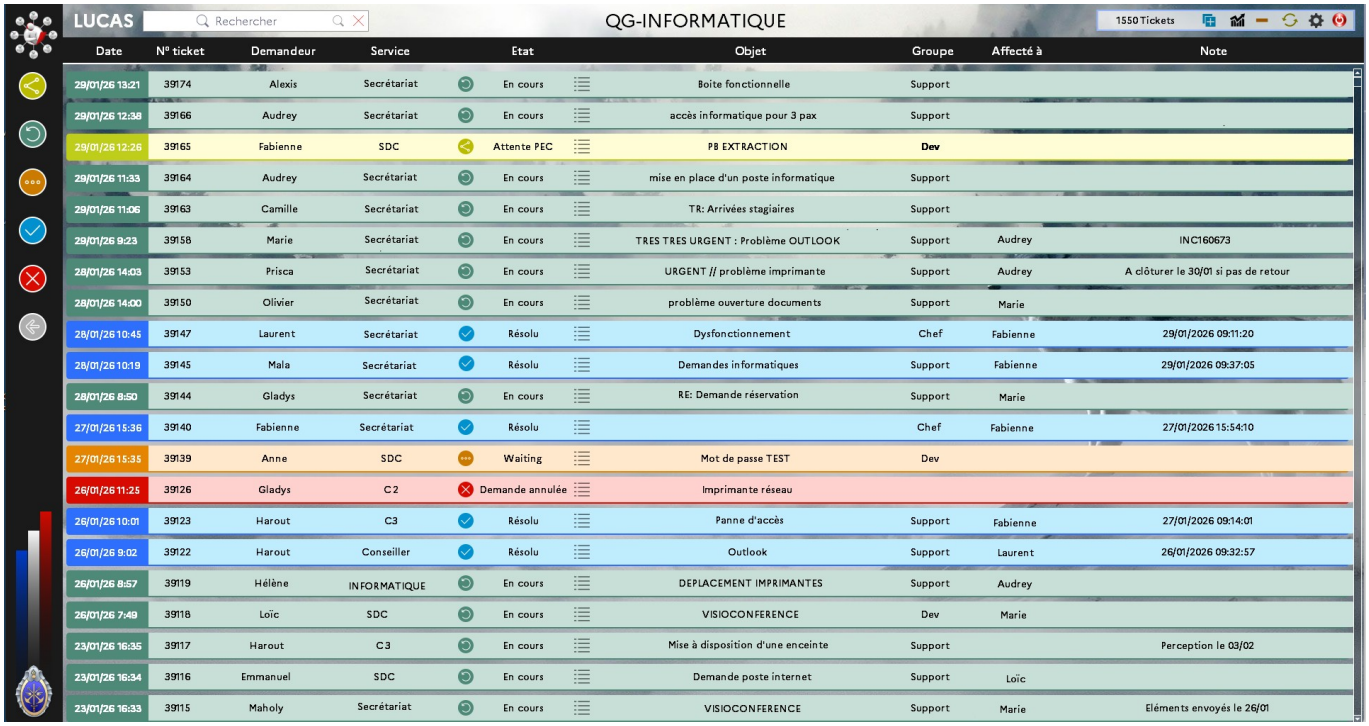
Les travaux se déroulent sur des machines virtuelles, ce qui permet de disposer d'environnements sécurisés et isolés, et de pouvoir tester différentes configurations sans impacter l'infrastructure principale. Cette méthode est particulièrement intéressante, car elle combine sécurité, flexibilité et autonomie pour les développeurs et testeurs.



Environement de travail

L'équipe m'a également présenté d'autres logiciels développés en interne, comme une application de gestion des décorations pour les civils et militaires, et un logiciel de gestion des incidents, similaire à GLPI[1].

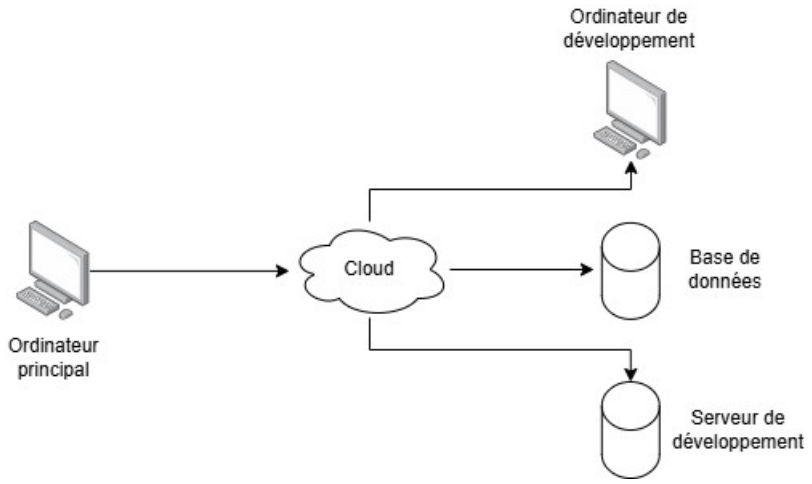
1 GLPI : Gestionnaire Libre de Parc Informatique



Date	N° ticket	Demandeur	Service	Etat	Objet	Groupe	Affecté à	Note
29/01/26 13:21	39174	Alexis	Secrétariat	En cours	Boite fonctionnelle	Support		
29/01/26 12:38	39166	Audrey	Secrétariat	En cours	accès informatique pour 3 pax	Support		
29/01/26 12:26	39165	Fabienne	SDC	Attente PEC	PB EXTRACTION	Dev		
29/01/26 11:33	39164	Audrey	Secrétariat	En cours	mise en place d'un poste informatique	Support		
29/01/26 11:06	39163	Camille	Secrétariat	En cours	TR: Arrivées stagiaires	Support		
29/01/26 9:23	39158	Marie	Secrétariat	En cours	TRES TRES URGENT : Problème OUTLOOK	Support	Audrey	INC160673
28/01/26 14:03	39153	Prisca	Secrétariat	En cours	URGENT // problème imprimante	Support	Audrey	A clôturer le 30/01 si pas de retour
28/01/26 14:00	39150	Olivier	Secrétariat	En cours	problème ouverture documents	Support	Marie	
28/01/26 10:45	39147	Laurent	Secrétariat	Résolu	Dysfonctionnement	Chef	Fabienne	29/01/2026 09:11:20
28/01/26 10:19	39145	Mala	Secrétariat	Résolu	Demandes informatiques	Support	Fabienne	29/01/2026 09:37:05
28/01/26 8:50	39144	Gladys	Secrétariat	En cours	RE: Demande réservation	Support	Marie	
27/01/26 15:36	39140	Fabienne	Secrétariat	Résolu		Chef	Fabienne	27/01/2026 15:54:10
27/01/26 15:35	39139	Anne	SDC	Waiting	Mot de passe TEST	Dev		
26/01/26 11:25	39126	Gladys	C2	Demande annulée	Imprimante réseau			
26/01/26 10:01	39123	Harout	C3	Résolu	Panne d'accès	Support	Fabienne	27/01/2026 09:14:01
26/01/26 9:02	39122	Harout	Conseiller	Résolu	Outlook	Support	Laurent	26/01/2026 09:32:57
26/01/26 8:57	39119	Hélène	INFORMATIQUE	En cours	DEPLACEMENT IMPRIMANTES	Support	Audrey	
26/01/26 7:49	39118	Loïc	SDC	En cours	VISIOCONFERENCE	Dev	Marie	
23/01/26 16:35	39117	Harout	C3	En cours	Mise à disposition d'une enceinte	Support		Perception le 03/02
23/01/26 16:34	39116	Emmanuel	SDC	En cours	Demande poste internet	Support	Loïc	
23/01/26 16:33	39115	Maholy	Secrétariat	En cours	VISIOCONFERENCE	Support	Marie	Eléments envoyés le 26/01

Logiciel de gestion des incidents

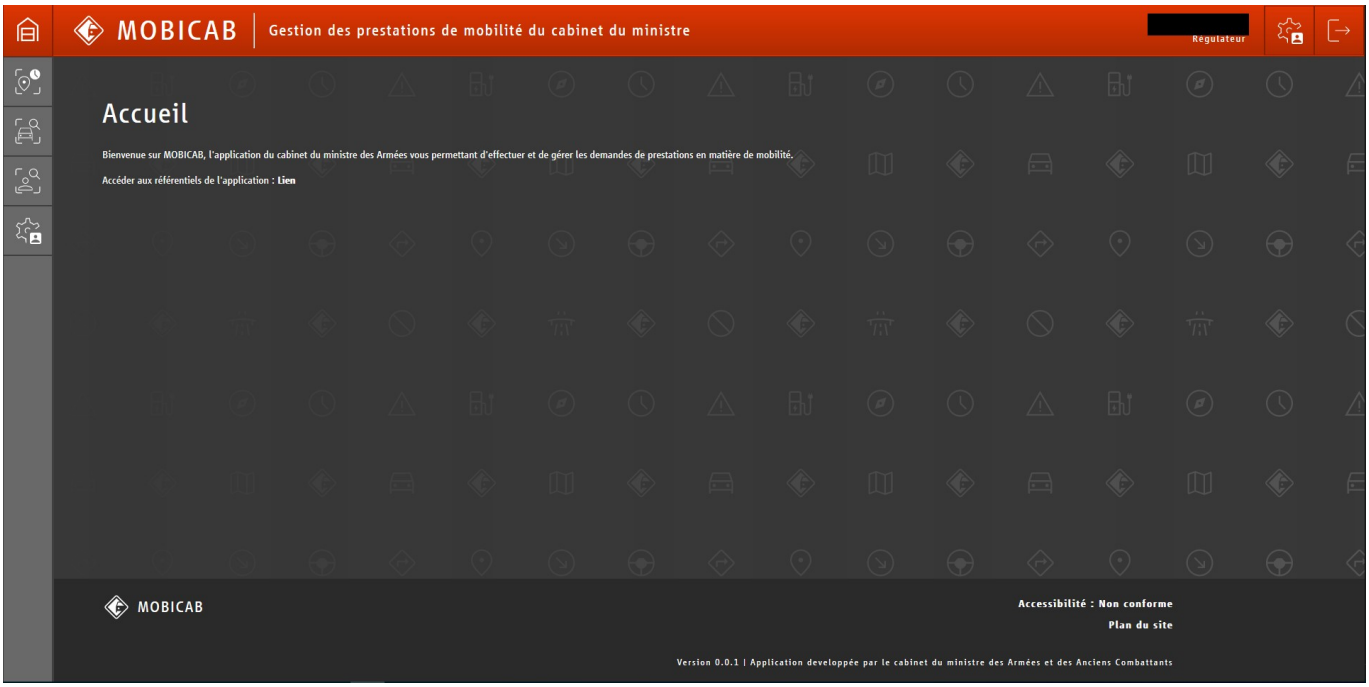
L'infrastructure repose sur un système cloud interne, qui joue le rôle de passerelle entre l'ordinateur de l'utilisateur et les environnements de développement. Grâce à ce cloud, chaque utilisateur peut accéder à une machine virtuelle de développement, sur laquelle le code est développé et testé de manière isolée. Le cloud permet également l'accès à la base de données et au serveur de développement, centralisant ainsi toutes les ressources nécessaires au développement.



Utilité du cloud dans le service

3. Le projet et l'application Mobicab

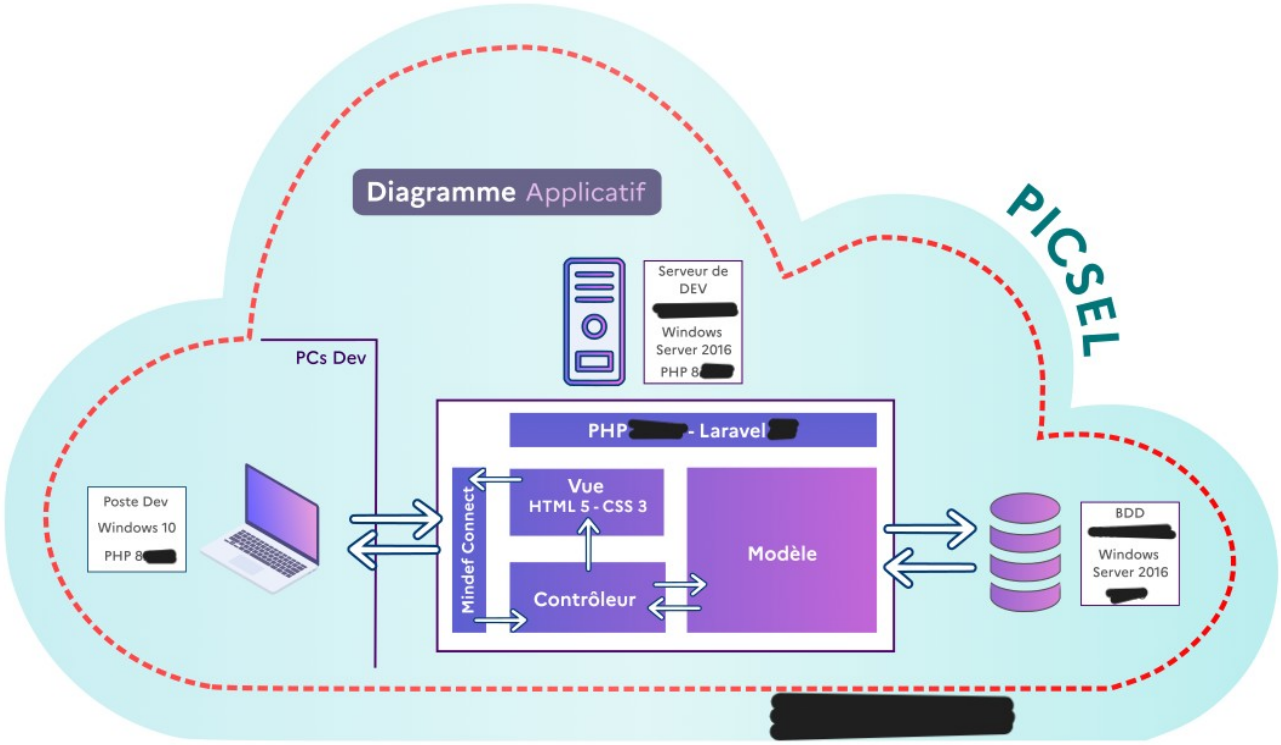
L'application principale sur laquelle j'interviens s'appelle Mobicab, un outil de gestion des parcs automobiles du cabinet. Elle est en cours de développement et est conçue avec Laravel et PHP 8.+, un choix technologique dicté par le cadre de cohérence technique du ministère des Armées.



Page d'accueil

Durant mon stage, l'équipe travaillait sur le sprint 3, un sprint dédié au développement de l'application. Les deux premiers sprints portaient sur l'analyse et la faisabilité du projet.

L'application et la base de données sont hébergées séparément. Le projet suit le modèle MVC (Modèle-Vue-Contrôleur) de Laravel, ce qui permet de séparer clairement la gestion des données, la logique métier et l'interface utilisateur.



Modèle MVC de l'application

Lors de mon arrivée, l'équipe travaillait sur la mise en conformité avec le RGAA[2], pour rendre l'application accessible à tous. J'ai pu observer l'utilisation d'attributs ARIA[3] et d'indices de tabulation (tab index) afin d'améliorer la navigation au clavier et l'accessibilité pour des utilisateurs ayant des besoins spécifiques.

J'ai aussi appris que les plans de site doivent inclure uniquement les pages principales et exclure les pages peu pertinentes, comme certaines pages CRUD[4].

2 RGAA : Référentiel Général d'Amélioration de l'Accessibilité
3 ARIA : Applications Internet Riches et Accessibles (Accessible Rich Internet Applications)
4 CRUD : Create, Read, Update, Delete (créer, lire, mettre à jour, supprimer)

4. Méthodes de travail

L'équipe travaille selon des méthodes agiles et utilise un Kanban physique, sous la forme d'un tableau réel avec des post-it. Chaque post-it correspond à un ticket, représentant une tâche à réaliser. Chaque ligne du tableau est associée à un développeur, et plusieurs colonnes permettent de suivre l'avancement du travail : une colonne Tasks, correspondant aux tâches à faire (équivalent d'un backlog), une colonne To Do pour les tickets attribués à un développeur, une colonne Doing regroupant les tickets en cours avec un pourcentage d'avancement, et enfin une colonne Done pour les tickets terminés.

Tableau Kanban utilisé par l'équipe de développement

Dans le cadre de mon stage, j'ai été positionné sur la dernière ligne du tableau, partagée avec un autre développeur, afin de faciliter le suivi de mon travail compte tenu de la durée limitée de mon stage (six semaines).

5. Développement et tests unitaires

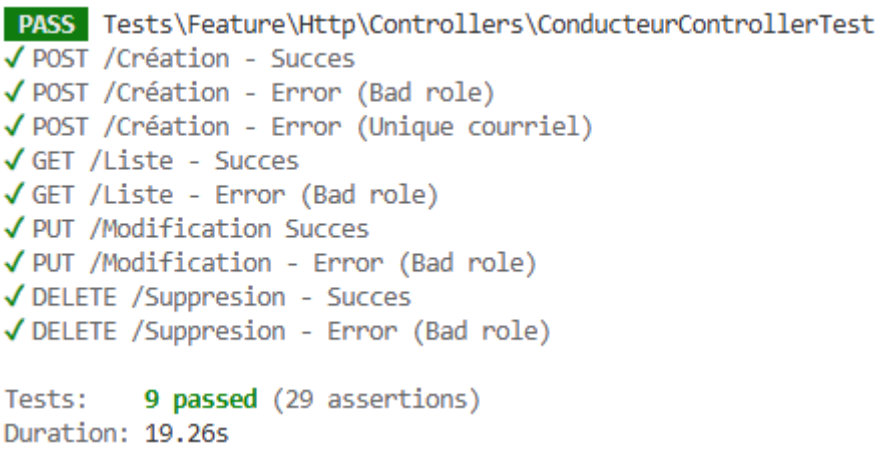
5.1 Premiers tests et table Conducteur

J’ai commencé par développer mes premiers tests unitaires, en m’appuyant sur des tests existants pour la table Conducteur. Les tests portaient sur :

- les expressions régulières ;
- la longueur des chaînes de caractères (maximum 50 caractères) ;
- les champs obligatoires.

J’ai découvert deux approches dans les tests :

- tester chaque colonne et chaque règle individuellement ;
- tester chaque colonne en regroupant toutes les règles dans un même test.



Tests d'intégration pour les conducteurs

5.2 Tests suivants et sur les autres tables

Pour la table Véhicule, aucun exemple n'était disponible. Je me suis inspiré de la table Conducteur pour créer des tests de type features (POST, PUT, DELETE).

```php
//
test(description: 'POST /Création - Succes', closure: function (): void {
    // Création d'un utilisateur avec le rôle approprié
    $user = Utilisateur::factory()->create([
        'util_role_id' => 1, // Rôle admin ou autre rôle avec les droits nécessaires
    ]);

    // Utilisation de la fonction validPayloadSuiviEntretien pour générer les données
    $payload = validPayloadSuiviEntretien();

    // Exécution de la requête
    $response = $this->actingAs(user: $user)
        ->post(uri: route(name: 'vehicules.entretiens.store'), data: $payload);

    // Vérification de la création dans la base de données
    $this->assertDatabaseHas(table: 'suivi_entretiens', data: [
        'suen_date_rdv' => date(format: "Y-m-d H:i", timestamp: strtotime(datetime: '20-01-2026 13:08')),
        'suen_obs' => 'Suivi OBS',
    ]);

    // Vérification du message flash
    $response->assertSessionHas(key: 'success', value: "L'entretien a été créé avec succès.");

    // Vérification que le suivientretien a bien été créé avec tous les champs
    $suivientretien = SuiviEntretien::where(column: 'suen_date_rdv', operator: date(format: "Y-m-d H:i", timestamp: strtotime(datet
        ->where(column: 'suen_obs', operator: 'Suivi OBS')
        ->first();
    $this->assertNotNull(actual: $suivientretien);
    $this->assertEquals(expected: 'Motif', actual: $suivientretien->suen_motif);
    $this->assertEquals(expected: 'Garage', actual: $suivientretien->suen_garage);
});
```

Test pour la création d'un enregistrement dans la table SuiviEntretien

La table Marque a servi de base pour tester la relation avec la table Modèle. J'ai rencontré des problèmes liés aux clés étrangères, que j'ai résolus en utilisant les Factory pour générer des données fictives et récupérer les identifiants corrects :

Marque::factory()->create()->marq_id

```php
<?php

namespace Database\Factories;

use App\Models\Modele;
use App\Models\Marque;
use Illuminate\Database\Eloquent\Factories\Factory;
use Illuminate\Support\Str;

0 references | 0 implementations
class ModeleFactory extends Factory
{
    /**
     * Le nom du modèle associé à la factory.
     *
     * @var string
     */
    0 references
    protected $model = Modele::class;

    /**
     * Définir les valeurs par défaut pour les attributs du modèle.
     *
     * @return array
     */
    0 references | 0 overrides
    public function definition(): array
    {
        return [
            'mode_libelle' => $this->faker->randomElement(array: ['Modele A', 'Modele B']),
            'mode_capacite' => $this->faker->randomElement(array: [0, 1, 2]),
            'mode_marq_id' => Marque::factory()->create()->marq_id,
            'mode_tyve_id' => $this->faker->randomElement(array: [1, 2, 3]),
        ];
    }
}
```

Factory pour la table Modele

Un autre problème était lié à la base de données de test, où les données accumulées provoquaient des conflits, notamment pour les adresses e-mail générées par Faker.

Erreur rencontrée due à la contrainte d'unicité des clés

Avec l'équipe, nous avons modifié le fichier TestCase pour réinitialiser certaines tables (Conducteur et Utilisateur) et éviter les doublons.

```php
protected function cleanDatabase(): void
{
    // Liste des tables à nettoyer
    $tables = [
        'utilisateurs',
        'vehicules',
        'entites',
        'conducteurs',
        'modeles',
        'marques',
        // Ajoutez ici les autres tables que vous voulez nettoyer
    ];

    // Utiliser la connexion de test explicitement
    $testDB = DB::connection(name: 'pgsql_test');

    foreach ($tables as $table) {
        if ($testDB->getSchemaBuilder()->hasTable(table: $table)) {
            $testDB->table(table: $table)->truncate();
        }
    }
}
```

Code permettant de réinitialiser les tables après le passage des tests

Lors du développement des tests pour la table Véhicule, j'ai également rencontré un problème concernant l'immatriculation des véhicules, qui devait rester unique. En réutilisant les mêmes méthodes que pour d'autres tables, le test ne passait pas et j'ai passé un certain temps à comprendre la cause. Avec l'aide de plusieurs membres de l'équipe, nous avons découvert que le problème venait du code principal, où la condition de contrainte unique sur le pattern du champ vehi_immat n'était pas correcte. Après correction, le test a passé avec succès.

Après ces corrections, j'ai pu finaliser la table Modèle et reprendre la table Véhicule, en appliquant les mêmes méthodes et en résolvant les dernières difficultés rencontrées.

J'ai ensuite travaillé sur la table Entités, ainsi que sur la table Utilisateur. Pour cette dernière, le patch ne passait pas initialement. Un développeur est venu m'aider à identifier le problème, car nous pensions que cela venait de mon programme. Après vérification, rien ne semblait incorrect. Nous avons alors testé directement sur l'application, en essayant de modifier un numéro de téléphone, et constaté que la modification ne fonctionnait pas.

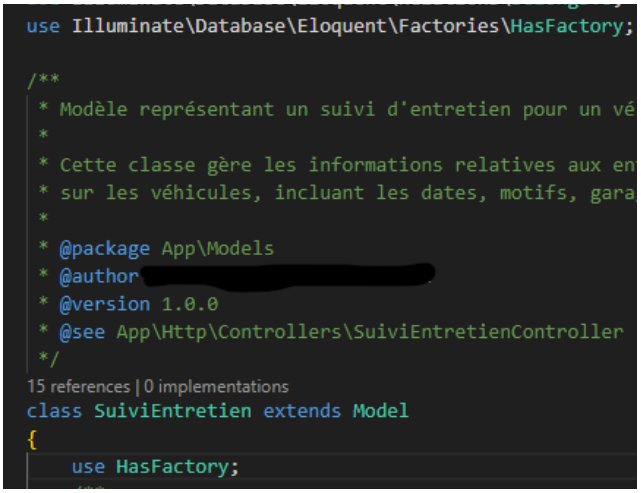
Nous avons séparément analysé le code front-end et découvert qu'un POST était utilisé alors qu'un PATCH était requis. Cette correction côté front a permis à l'action de fonctionner. Du côté des tests, le problème provenait du fichier UtilisateurRequest : l'identifiant était exigé pour tous les cas de figure, alors qu'il n'était valide que pour la modification d'un numéro de téléphone. Nous avons donc séparé les responsabilités en deux fichiers :

- UtilisateurRequest pour toutes les actions liées aux utilisateurs ;
- MonCompteRequest pour les modifications de compte effectuées par les utilisateurs eux-mêmes (comme le changement de numéro de téléphone).

Grâce à ces corrections, les tests ont pu être exécutés avec succès sur la table Utilisateur, et les problèmes précédemment rencontrés ne se sont plus reproduits.

Pour finir, j'ai travaillé sur la table Suivi d'entretien, dernière table sur laquelle j'ai développé des tests.

J'y ai rencontré principalement deux problèmes. Le premier concernait l'utilisation des Factory : j'avais créé une factory, mais mes tests ne la prenaient pas en compte. Après analyse, j'ai compris que cela venait du fait que le modèle SuiviEntretien n'incluait pas le trait HasFactory, ce qui empêchait Laravel de reconnaître la factory associée. Une fois ce trait ajouté, les tests ont pu utiliser correctement la factory.



```
use Illuminate\Database\Eloquent\Factories\HasFactory;

/**
 * Modèle représentant un suivi d'entretien pour un vé
 *
 * Cette classe gère les informations relatives aux en
 * sur les véhicules, incluant les dates, motifs, gara
 *
 * @package App\Models
 * @author [redacted]
 * @version 1.0.0
 * @see App\Http\Controllers\SuiviEntretienController
 */
15 references | 0 implementations
class SuiviEntretien extends Model
{
    use HasFactory;
```

HasFactory ajouté pour la table SuiviEntretien

Le second problème était lié aux IDs dans les tests d'intégration.

```
SQLSTATE[22P02]: Invalid text representation: 7 ERREUR:  syntaxe en entrée invalide pour le type bigint : «  »
CONTEXT:  paramètre de portail non nommé $1 = '' (Connection: pgsql_test, SQL: select * from "vehicules" where "vehicul
es"."vehi_id" =  limit 1)

  at tests\Feature\Http\Controllers\SuiviEntretienControllerTest.php:125
    121         ->get(route('vehicules.entretiens.show', (int)$id_vehi));
    122         // dd(route('vehicules.entretiens.show', (int)$id_vehi));
    123     // dd($response);
    124     // Vérifier la réponse
  ➜ 125     $response->assertStatus(200)
    126             ->assertViewIs(route('vehicules.entretiens.show', (int)$id_vehi));
    127
    128 });
    129


  Tests:    1 failed, 7 passed (20 assertions)
  Duration: 8.41s
```

Erreur d'identifiants non reconnus

Pour que les requêtes POST fonctionnent, les IDs devaient être encodés en Base64. En utilisant les IDs récupérés comme pour les autres tables, les tests échouaient sans message

d'erreur clair. Après avoir appliqué l'encodage Base64 aux IDs dans les tests, les requêtes ont fonctionné comme attendu.

```php
// Faire la requête pour la liste
$response = $this->actingAs(user: $user)
    ->get(uri: route(name: 'vehicules.entretiens.show', parameters: ['vehi_id' => base64_encode(string: $id_vehi)]));
    // dd(route('vehicules.entretiens.show', (int)$id_vehi));
// dd($response);
// Vérifier la réponse
$response->assertStatus(status: 200);
```

Encodage Base64 ajouté pour récupérer un identifiant valide

Ces deux corrections m'ont permis de finaliser le développement des tests pour cette table et de conclure cette phase sur toutes les tables concernées.

```
PASS  Tests\Feature\Http\Controllers\VehiculeControllerTest
✓ POST /Création - Succes
✓ POST /Création - Error (Bad role)
✓ POST /Création - Error (Renseigner kilometrage)
✓ POST /Création - Error (Unique immat)
✓ GET /Liste - Succes
✓ GET /Liste - Error (Bad role)
✓ PUT /Modification Succes
✓ PUT /Modification - Error (Bad role)
✓ DELETE /Suppresion - Succes
✓ DELETE /Suppresion - Error (Bad role)

Tests:    611 passed (728 assertions)
Duration: 122.45s
```

L'ensemble des tests passent

5.3 Tests SSO sur l'application

Pour la connexion à l'application, un système de Single Sign-On (SSO) est utilisé. Cela permet aux utilisateurs de se connecter à l'application avec les mêmes informations d'identification que celles qu'ils utilisent pour accéder à leurs poste de travail. Le SSO a donc été ajouté à l'application, mais il restait encore à effectuer les tests d'intégration. J'ai donc dû comprendre le fonctionnement de ce SSO ainsi que du programme qui lui était associé.

```php
/**
 * Met à jour un utilisateur s'il existe, sinon le crée.
 *
 * @param array $data Les données du utilisateur
 * @return Utilisateur L'utilisateur créé ou mis à jour
 */
4 references | 0 overrides
public function upsertUtilisateur(string $courriel, string $nom, string $prenom, string $login, ?string $mobile = null): Utilisateur
{
    $data = [
        'util_courriel' => $courriel,
        'util_nom' => strtoupper(string: $nom),
        'util_prenom' => $prenom,
        'util_login' => $login
    ];
    if ($this->utilisateurExistsByCourriel(courriel: $courriel)) {
        $utilisateur = Utilisateur::where(column: 'util_courriel', operator: $courriel)->first();

        if ($mobile !== null && $mobile !== '') {
            $data['util_tel'] = $mobile;
        }
        return $this->updateUtilisateur(utilisateur: $utilisateur, data: $data);
    }
    $data['util_etat'] = 1;
    return $this->createUtilisateur(data: $data);
}
```

Programme de connexion par SSO

Le programme nécessitait donc deux tests : le premier pour simuler la première connexion d'un utilisateur, le second pour simuler le cas où l'utilisateur existe déjà en base de données.

Lors de la première connexion, il est nécessaire de créer l'utilisateur dans la base de données afin qu'il puisse se connecter.

```php
//
// 🚀 CRÉATION - A la connexion créer un utilisateur si il n'existe pas
//
test(description: 'POST /Création au login - Succes', closure: function (): void {
    // Création d'un utilisateur avec le rôle approprié
    $user = Utilisateur::factory()->create([
        'util_role_id' => 1, // Rôle admin ou autre rôle avec les droits nécessaires
    ]);

    // Utilisation de la fonction validPayloadUtilisateur pour générer les données
    $payload = validPayloadUtilisateur();

    $utilServ = new UtilisateurService();
    $response = $utilServ->upsertUtilisateur(courriel: $payload['util_courriel'],nom: $payload['util_nom'],pr

    // Vérification de la création dans la base de données
    $this->assertDatabaseHas(table: 'utilisateurs', data: [
        'util_prenom' => 'Jean',
        'util_login' => 'jean.dupont',
    ]);

    // Vérification que l'entite a bien été créé avec tous les champs
    $utilisateur = Utilisateur::where(column: 'util_nom', operator: 'DUPONT')
        ->where(column: 'util_login', operator: 'jean.dupont')
        ->first();
    $this->assertNotNull(actual: $utilisateur);
    $this->assertEquals(expected: 'Jean', actual: $utilisateur->util_prenom);
    $this->assertEquals(expected: 'dupont.jean@intradef.gouv.fr', actual: $utilisateur->util_courriel);
    $this->assertEquals(expected: 1, actual: $utilisateur->util_etat);
    $this->assertEquals(expected: 4, actual: $utilisateur->util_role_id);
});
```

Test d'intégration pour la création d'un utilisateurà la première connexion

Pour les connexions suivantes, l'utilisateur est déjà enregistré ; lorsqu'il se connecte, comme le montre le programme, ses informations peuvent alors être modifiées.

```php
//
// Mise à jour - A la connexion modifier un utilisateur si il existe déjà
//
test(description: 'PUT /Modification au login - Succes', closure: function (): void {
    // Création d'un utilisateur avec le rôle approprié
    $user = Utilisateur::factory()->create([
        'util_role_id' => 1, // Rôle admin ou autre rôle avec les droits nécessaires
    ]);

    // Utilisation de la fonction validPayloadUtilisateur pour générer les données
    $payload = validPayloadUtilisateur();

    $utilServ = new UtilisateurService();
    $response0 = $utilServ->upsertUtilisateur(courriel: $payload['util_courriel'],nom: $payloa


    // Vérification de la création dans la base de données
    $this->assertDatabaseHas(table: 'utilisateurs', data: [
        'util_tel' => null,
        'util_login' => 'jean.dupont',
    ]);

    $payload = validPayloadUtilisateur(override: [
        'util_tel' => '0600000000'
    ]);
    $response = $utilServ->upsertUtilisateur(courriel: $payload['util_courriel'],nom: $payload

    // Vérification de la création dans la base de données
    $this->assertDatabaseHas(table: 'utilisateurs', data: [
        'util_tel' => '0600000000',
        'util_login' => 'jean.dupont',
    ]);
});
```

Test d'intégration pour la modification d'un utilisateur lors d'une connexion

Ces deux tests ont ainsi été réalisés et validés.

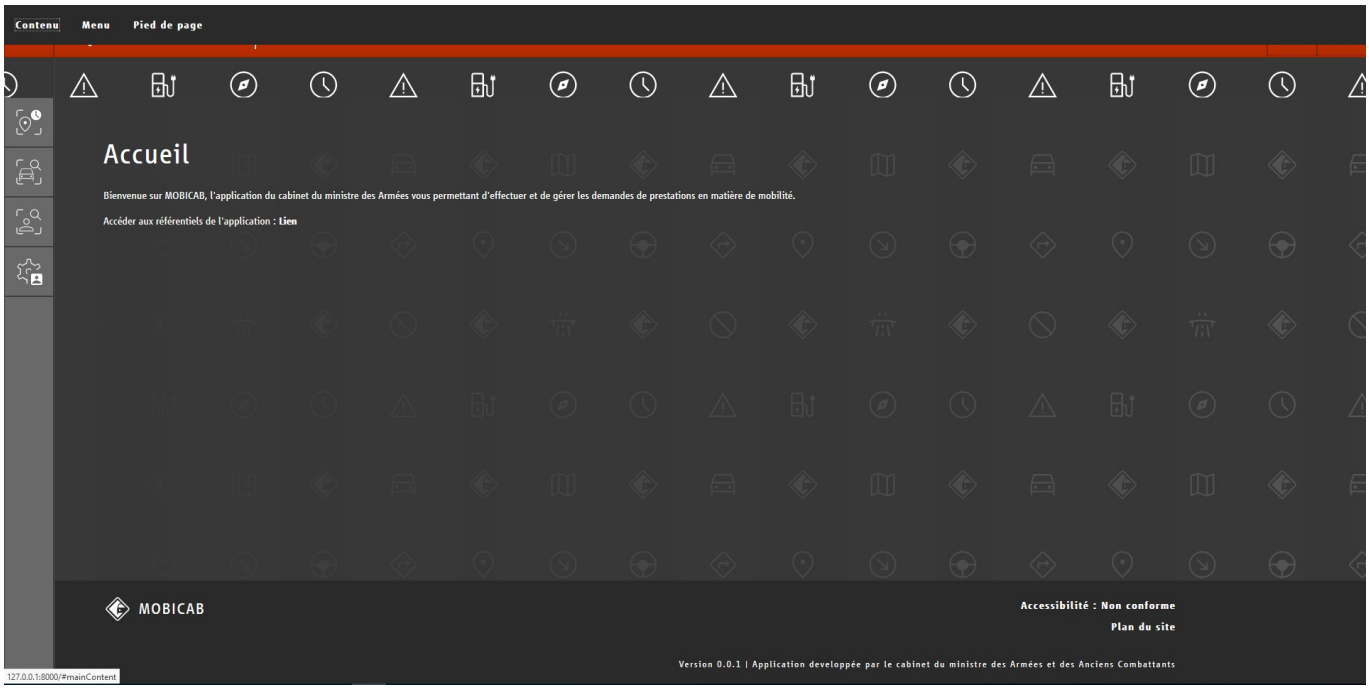
```
✓ POST /Création au login - Succes
✓ PUT /Modification au login - Succes
```

Tests d'intégration réussi

6. Front

6.1 Correction d’un problème de superposition des liens d’évitement (accessibilité)

Un problème d'accessibilité m'a été signalé concernant les liens d'évitement : lorsqu'ils étaient activés, ceux-ci passaient par-dessus le header de la page. Il m'a été indiqué que ce comportement n'était pas présent auparavant et qu'il était apparu récemment.

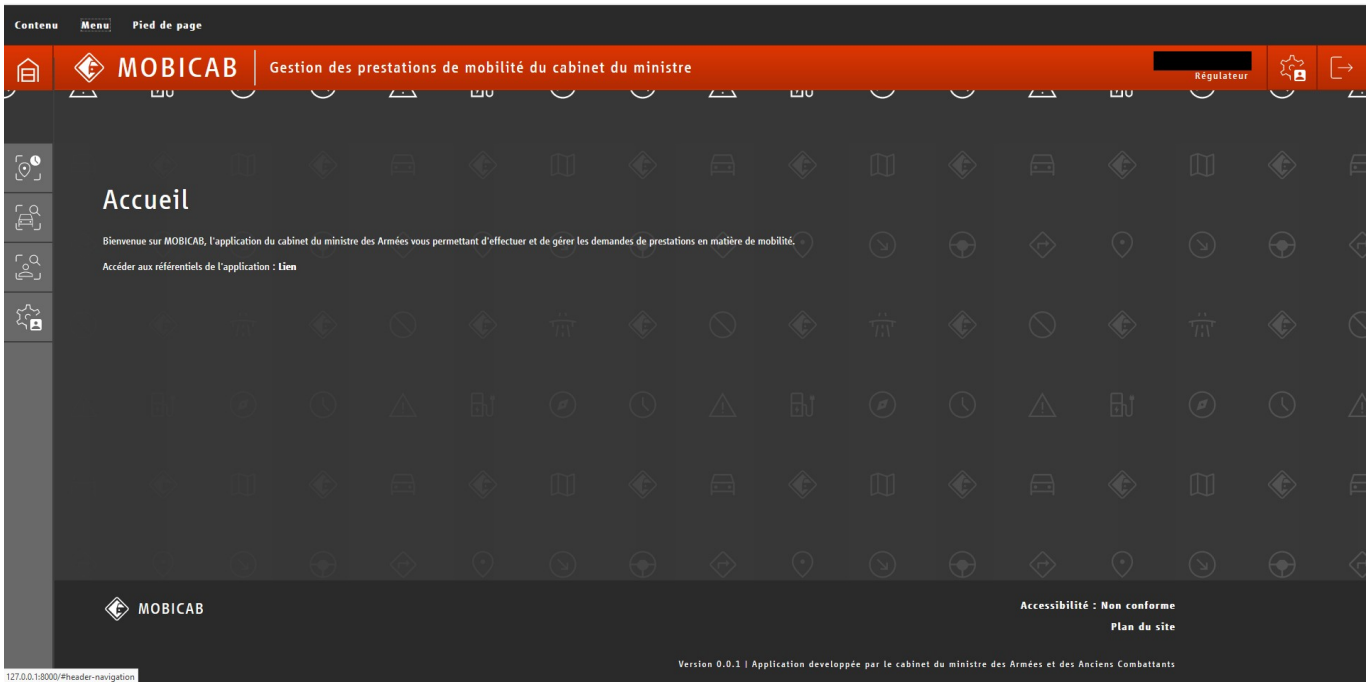


Problème initial

J'ai commencé par analyser la structure et le fonctionnement du header, tout en examinant l'historique des modifications (pushs) réalisés par les autres développeurs. Cette analyse m'a permis d'identifier une modification récente : la position du header avait été changée de *position: relative* à *position: fixed*.

```css
.header-container {
    position: relative; /* anciennement fixed */
    width: 100%;
    display: flex;
    flex-direction: row;
    overflow: hidden;
    margin: 0;
    top: 0;
    z-index: 100;
}
```

La position fixed passé à relative

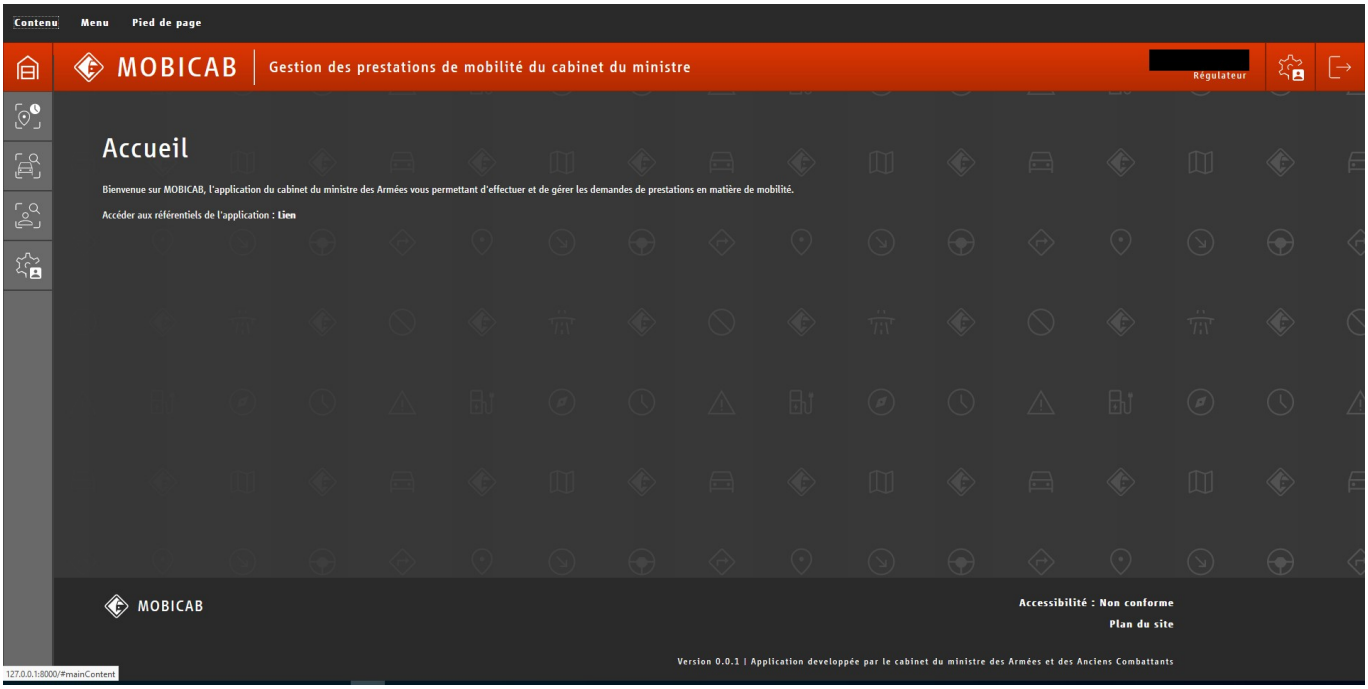


Accueil après correction de la position

En rétablissant la position relative, le problème de superposition a été corrigé. Cependant, un espace d'environ 74 pixels est alors apparu entre le header et la sidenav (barre de navigation latérale). Après analyse du code et des feuilles de style, j'ai identifié cette marge dans le fichier style.css. Sa suppression a permis de rétablir un affichage correct. Le problème des liens d'évitement a ainsi été entièrement résolu.

```css
.body-container {
    height: calc(100vh - 70px);
    width: 100%;
    display: flex;
    flex-direction: row;
    overflow: hidden;
    background-image: linear-gradient(to left, rgba(55, 55, 55, 0.85), rgba(55, 55, 55, 1));
    flex-grow: 1;
    /* margin-top: 74px; */
}
```

Margin-top retiré



Accueil avec problème liens d'évitement corrigé

Le problème des liens d'évitement a ainsi été entièrement résolu.

6.2 Problème de gestion du focus clavier sur la sidenav

La barre de navigation latérale (sidenav) dispose, comme le reste du site, d'un système de navigation au clavier par tabulation. Un problème d'accessibilité a été identifié : lorsque l'utilisateur atteignait la fin de la tabulation dans la sidenav, le focus continuait à parcourir des éléments appartenant à des menus pourtant fermés, au lieu de passer directement au contenu principal de la page.

L'origine du problème venait du fait que les menus de la sidenav étaient simplement réduits à une largeur de 0 px pour être dissimulés. Malgré cela, ces éléments restaient accessibles au focus clavier, ce qui ne respecte pas les recommandations du RGAA et provoquait une navigation plus difficile lors de l'utilisation de la tabulation.

Dans un premier temps, j'ai tenté de rendre ces éléments inaccessibles en utilisant les attributs *aria-hidden* et *tabindex="-1"*. Toutefois, cette solution n'a pas suffi à empêcher complètement la navigation par tabulation. J'ai ensuite utilisé la propriété CSS[5] *display: none*, ce qui a permis de corriger le problème de focus, mais a entraîné la suppression des animations d'ouverture et de fermeture des menus.

5 CSS : Cascading Style Sheets

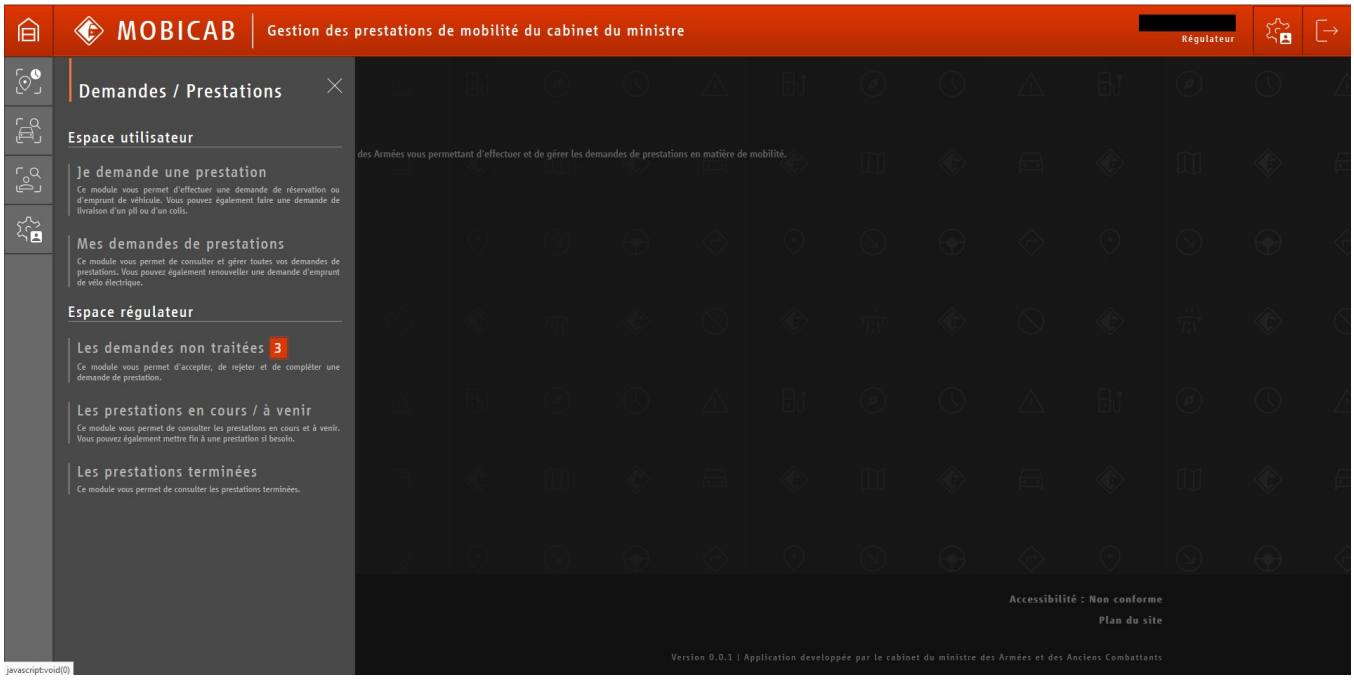
```
// Fonction pour cacher totalement l'élément de side nav
async function setHidden(sidenavId) {
        document.getElementById(id).setAttribute('style', 'transition-behavior: allow-discrete;');
        const sidenav = document.getElementById(sidenavId);
        sidenav.setAttribute('aria-hidden', 'true');
        sidenav.style.display = 'none';
}

// Retirer ce qui cache les éléments de la side nav
async function removeHidden(sidenavId) {
        document.getElementById(id).setAttribute('style', 'transition-behavior: allow-discrete;');
        const sidenav = document.getElementById(sidenavId);
        sidenav.setAttribute('aria-hidden', 'false');
        sidenav.style.display = '';
}
```

Fonctions JavaScript pour masquer les éléments de la barre latérale

Après échanges avec un développeur de l'équipe, nous avons décidé de changer d'approche. Plutôt que de masquer complètement les menus, j'ai revu la logique de navigation au clavier afin de faire passer directement le focus de la fin de la sidenav vers les éléments du contenu principal, sans traverser les menus fermés. Cette solution a permis de conserver les animations existantes tout en assurant une navigation cohérente et conforme aux règles d'accessibilité.

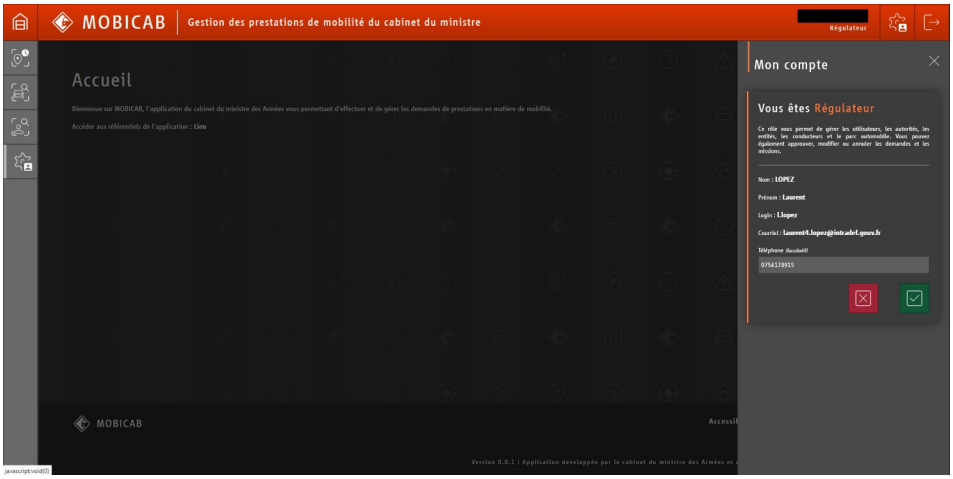
6.3 Sidenav déplacée à droite pour le menu « Mon compte »

Dans l'application, tous les menus de la sidenav sont initialement positionnés à gauche :



Menu de la sidenav à gauche

Étant donné qu'un bouton « Mon compte » avait été créé initialement sur l'application (à côté du bouton de déconnexion), il m'a été demandé de déplacer le menu vers la droite afin de correspondre à ce bouton. Après quelques modifications en CSS, j'ai obtenu un résultat conforme aux attentes. Pour y parvenir, j'ai ajouté une classe CSS au menu « Mon compte » afin de positionner la sidenav à droite.



Menu de la sidenav déplacé à droite

```css
.sidenav {
    height: calc(100vh - 70px);
    width: 0px;
    position: fixed;
    top: 74px;
    left: 70px;
    border-style: solid;
    border-width: 1px;
    border-color: #3f3f3f;
    background-color: #494949;
    overflow-x: hidden;
    overflow-y: auto;
    white-space: nowrap;
    transition: 0.5s;
}

.sidenav a {
    padding: 5px 5px 5px 20px;
    text-decoration: none;
    font-family: 'Vialog LT Regular';
    font-size: 22px;
    letter-spacing: 0.1rem;
    color: #b4b4b4;
    display: block;
    transition: 0.5s;
}

.sidenav-right {
    position: fixed;
    right: 0px ;
    left: auto;
}
```

Classe CSS ajoutée : sidenav-right

6.4 Mise en place d'une page d'accueil responsive

La page d'accueil de l'application présentait plusieurs problèmes de responsive. Il a été demandé d'adapter l'affichage des cartes afin qu'elles s'organisent par trois par ligne sur des écrans de taille intermédiaire, puis par deux ou une selon la largeur de l'écran et le support utilisé.

Accueil de base sans responsive

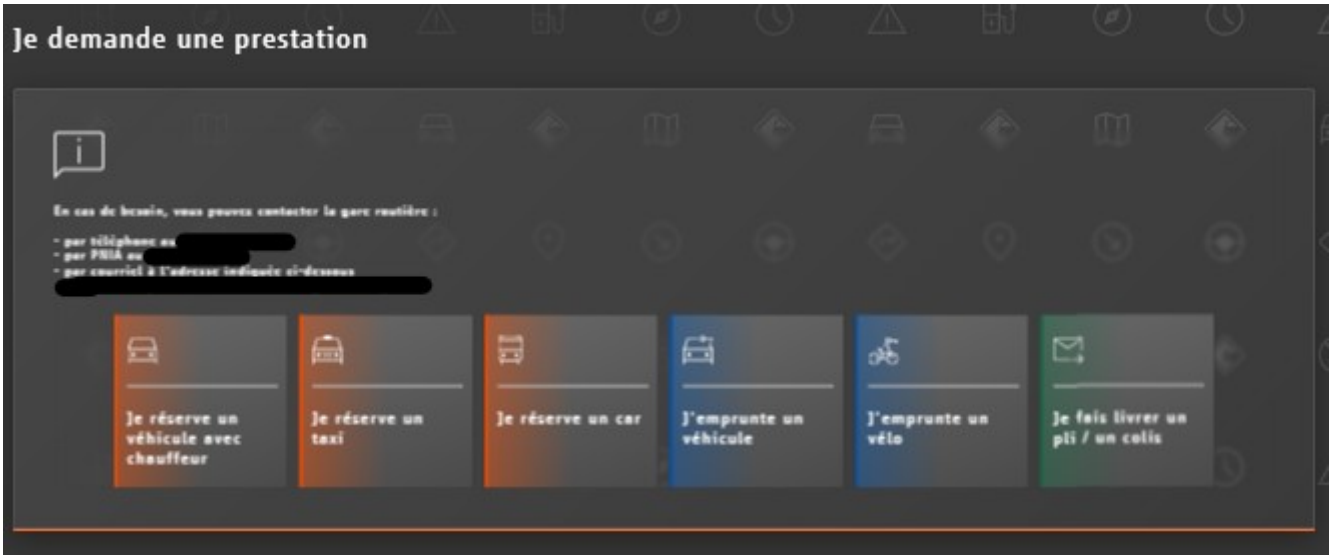
Pour répondre à cette demande, la mise en page a été revue en utilisant les Flexbox. Cette approche permet une adaptation plus souple de l'affichage selon la taille de l'écran.

Cette modification a nécessité la suppression de certaines propriétés width définies avec des valeurs fixes. Celles-ci empêchaient le redimensionnement correct des cartes et provoquaient des problèmes d'affichage sur les écrans plus petits.

Une fois ce premier point corrigé, un dernier problème a été identifié : dans certains formats d'écran, le flou appliqué en arrière-plan du conteneur se superposait aux cartes, altérant leur lisibilité. Ce problème ayant déjà été rencontré lors d'un projet personnel antérieur, une solution a pu être mise en place rapidement.

```
.tiles-columns-accueil {
+    flex-wrap: wrap;
     width: auto;
     height: 100%;
-    display: inline-flex;
+    display: flex;
     flex-direction: row;
     margin-left: 40px;
     margin-right: 40px;
     margin-top: 40px;
     margin-bottom: 40px;
+    justify-content: space-between;
}

.text-area-accueil {
-    height: 100%;
+    /* height: 100%; */
     display: block;
-    margin-right: 20px;
+    /* margin-right: 20px; */
+    margin-right: 40px;
     text-align: justify;
     text-justify: inter-word;
+    white-space: normal;
}

.text-area-accueil span {
-    width: 400px !important;
+    /* width: 400px !important; */
     display: inline-block;
     font-family: 'Vialog LT Regular
     font-size: 13px;
     font-weight: normal;
     letter-spacing: 0.1rem;
     color: #ffffff;
+    overflow-wrap: break-word;
+    white-space: normal;
}
```

Modifications CSS pour le responsif



Problème de flou

Le flou a ainsi été appliqué via une div dédiée, distincte du contenu textuel et des cartes. Cette séparation a permis de conserver l'effet visuel souhaité tout en évitant toute interaction indésirable entre le flou de fond et les éléments affichés au premier plan.

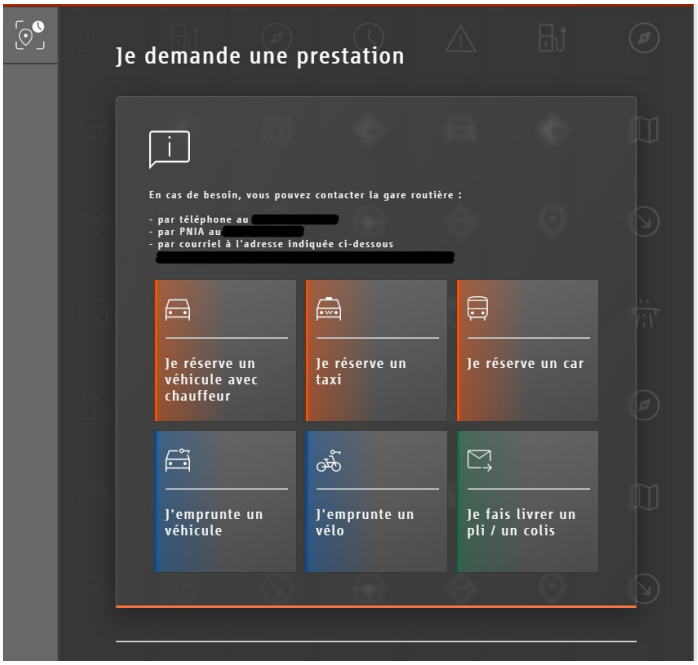
```
 .tiles-container-accueil {
-    min-width: fit-content;
+    /* min-width: fit-content; */
     height: auto;
     display: inline-flex;
     flex-direction: column;
@@ -1537,53 +1537,83 @@ textarea {
     border-bottom: 3px solid #fd7841;
     border-radius: 5px 5px 0px 0px;
     background: linear-gradient(135deg, rgba(255, 255, 255, 0.08), rgba(255, 255, 255, 0.02));
+    /* backdrop-filter: blur(2.5px);
+    -webkit-backdrop-filter: blur(2.5px); */
+    box-shadow: 0 8px 32px 0 rgba(0, 0, 0, 0.37);
+    position: relative;
+}
+
+.blur-class {
+    width: 100%;
+    height: 100%;
+    position: absolute;
     backdrop-filter: blur(2.5px);
     -webkit-backdrop-filter: blur(2.5px);
-    box-shadow: 0 8px 32px 0 rgba(0, 0, 0, 0.37);
+    z-index: -1;
+}
+
+.d-flex {
+    display: flex;
 }
```

CSS pour flou séparé

```
<div class="tiles-container-accueil">
    <div class="blur-class"></div>
    <div class="tiles-columns-accueil">
        <div class="text-area-accueil">
            <img src="{{ asset(path: 'img/icons/information.svg') }}" alt="Information" title="Information" height="50px" width="50px"><br aria-hidden="true" />
```

Div flou séparé

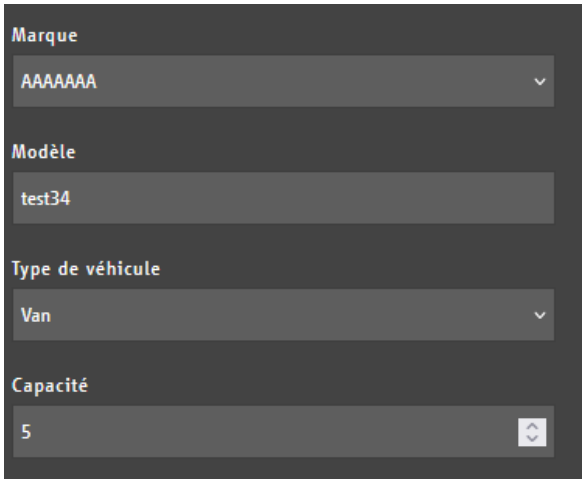
L'affichage est désormais correct sur l'ensemble des formats d'écran testés.



Accueil responsive

6.5 Désactiver des champs dans un formulaire

Dans le formulaire des véhicules, il a été nécessaire de désactiver certaines listes déroulantes, comme celles correspondant à la marque ou au type. L'objectif était d'empêcher l'utilisateur de modifier la marque d'un véhicule ou de changer son type, par exemple passer d'une voiture à un vélo.



Champs avant d'être désactivé

J'ai d'abord essayé de mettre simplement l'attribut disabled sur ces champs. Cependant, les tests ont montré que les règles de validation Laravel ne prenaient pas en compte les champs désactivés. Les valeurs de ces listes n'étaient donc pas envoyées lors de la soumission du formulaire, ce qui empêchait leur utilisation côté serveur.

```
1            @csrf
2            @method('PUT')
3            <div class="inputs-area">
4  +             <x-forms.input-hidden name="mode_id" value="{{ $modele->mode_id }}" />
5  +             <x-forms.input-hidden name="mode_marq_id" value="{{ old('mode_marq_id', $modele->mode_marq_id) }}" />
6  +             <x-forms.input-hidden name="mode_tyve_id" value="{{ old('mode_tyve_id', $modele->mode_tyve_id) }}" />
7  +             <x-forms.input-text label="Marque" name="mode_marq_id" placeholder="Choisir une marque" :options="$marques" :col
   umns="['marq_id', 'marq_libelle']" value="{{  $modele->marque->marq_libelle }}" required disabled />
8              <x-forms.input-text label="Modèle" name="mode_libelle" placeholder="Renseigner le modèle" value="{{ old('mode_li
   belle', $modele->mode_libelle) }}" required />
9  +             <x-forms.input-text label="Type de véhicule" name="mode_tyve_id" placeholder="Choisir un type de véhicule" :opti
   ons="$typesVehicule" :columns="['tyve_id', 'tyve_libelle']" value="{{ $modele->typeVehicule->tyve_libelle }}" required disable
   d />
0              <x-forms.input-number label="Capacité" name="mode_capacite" placeholder="Renseigner la capacité" value="{{ old
   ('mode_capacite', $modele->mode_capacite) }}" required />
1            </div>
2            <div class="form-buttons-area">
```

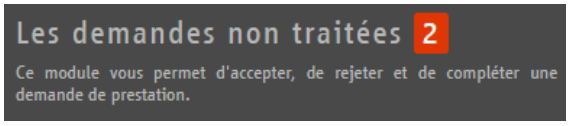
Code modifié

Pour résoudre ce problème, j'ai gardé les champs désactivés pour l'utilisateur, tout en créant des champs cachés contenant les mêmes valeurs. Ces champs cachés sont soumis avec le formulaire, ce qui permet de conserver les données et de les utiliser pour les validations et l'enregistrement.

Champs désactivé

6.6 Animations d'une notification à retirer

Dans l'application, un menu de la barre latérale affichait une notification avec une animation. Cette animation n'était pas adaptée aux utilisateurs épileptiques et il a été décidé de la retirer pour améliorer l'accessibilité.



La notification

Pour cela, j'ai commencé par retrouver le code CSS lié à la pop-up, puis j'ai simplement retiré l'animation. Cette modification a permis de conserver la notification visible sans effet visuel gênant.

```css
.pulse-notification {
    display: inline-block;
    background: #df3603;
    border-radius: 3px;
    font-size: 22px;
    font-weight: normal;
    color: #ffffff;
    padding: 0.13rem 0.35rem;
    position: relative;
}

.pulse-notification::before {
    content: '';
    display: block;
    position: absolute;
    top: 0;
    left: 0;
    right: 0;
    bottom: 0;
    border-radius: 3px;
    /* animation: pulse 1s ease infinite; */
    border: 3px double #df3603;
}
```

Code CSS corrigé

Il m'a également été demandé de cacher la pop-up lorsque le nombre de notifications est nul. Pour cela, j'ai ajouté une condition Laravel dans le template Blade :

```
@if ($count>=1)<div class='pulse-notification' @class(array: ['subtitle'])>{{ $count }}</div> @endif
```

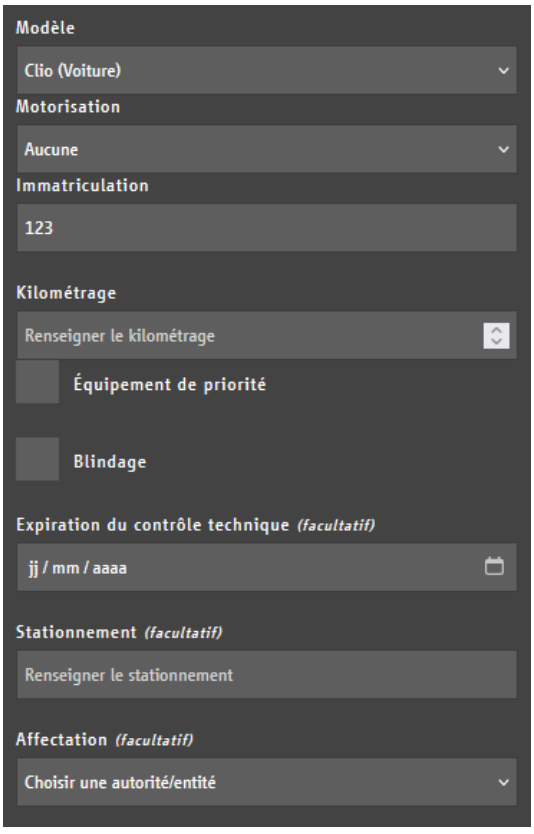
Condition ajoutée

Cette approche permet que la pop-up n'apparaisse que lorsqu'il y a réellement des notifications, évitant ainsi un affichage inutile et améliorant l'expérience utilisateur.

Notification retirée

6.7 Formulaire dynamique pour le kilométrage des véhicules

Dans l'application, lors de la création d'un véhicule, il n'était pas nécessaire de renseigner le kilométrage pour les vélos. Il a donc fallu rendre le formulaire un peu dynamique, afin que ce champ soit masqué lorsque l'utilisateur choisit un modèle de type vélo.



Formulaire avec kilométrage

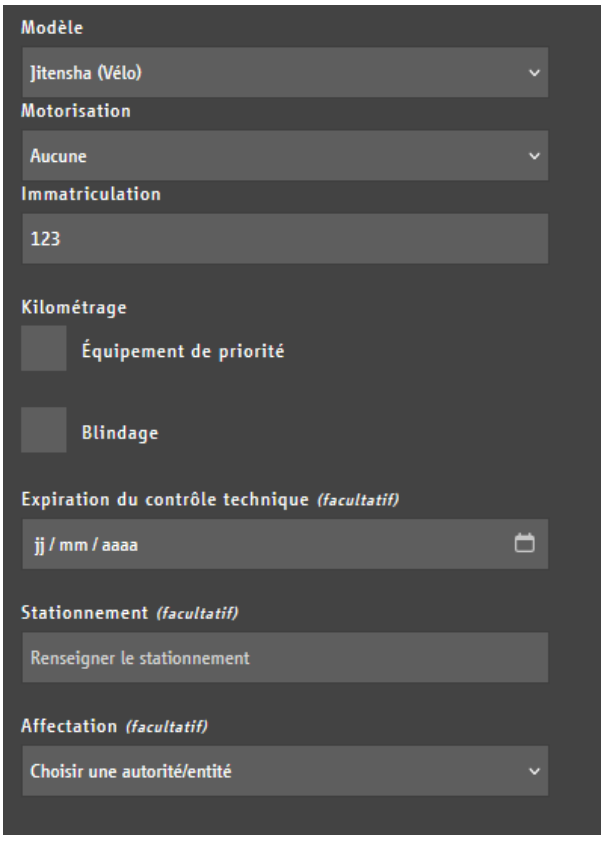
Pour cela, j'ai utilisé du JavaScript, un script qui détecte le changement dans le select correspondant au modèle du véhicule, puis vérifie si l'option sélectionnée contient un vélo. Si

c'est le cas, le champ kilométrage est caché, sinon il reste visible pour les autres types de véhicules.

```
@push('scripts')
<script>

    document.addEventListener("DOMContentLoaded", suiviKiloGestion());
    const modele = document.getElementById('vehi_mode_id');
    modele.addEventListener("change", suiviKiloGestion);

    function suiviKiloGestion(){
        console.log('test');
        const suivi_kilo = document.getElementById('suki_kilo_nve');
        const modele = document.getElementById('vehi_mode_id');
        var option = modele.options[modele.selectedIndex].text;
        if (option.includes('Vélo')) {
            suivi_kilo.style.display = "none";
            suivi_kilo.value = 1;
        } else {
            suivi_kilo.style.display = "";
            suivi_kilo.value= "";
        }
    }

</script>
@endpush
```

Code javascript pour le formulaire dynamique

Cette solution permet de simplifier le formulaire, d'éviter que des informations inutiles soient saisies pour certains véhicules, et d'améliorer l'expérience utilisateur.



Formulaire sans kilométrage (pour vélos)

6.8 Textareas et règles de validation

Dans l'application, certains textareas n'étaient pas correctement pris en compte par le système de validation Laravel. Même lorsqu'ils étaient marqués comme required, leur contenu n'était pas envoyé ou validé correctement.

Pour corriger ce problème, j'ai commencé par adapter le template des textareas pour qu'ils prennent en compte les attributs required et disabled lorsqu'ils étaient présents. Cela permet de gérer côté interface ce qui est obligatoire ou non.

```
<textarea oninput="auto_grow(this)" name="{{ $name }}" placeholder="{{ $placeholder }}" maxlength="{{ $max }}" class="@error($name) is-invalid @enderror"
    @if(!empty($required) || $required === true) required='true' @endif
    @if(!empty($disabled) || $disabled === true) disabled='true' @endif>{!! old(key: $name, default: $value) !!}</textarea>
```

Code modifié

Ensuite, il a fallu modifier la rule correspondante dans Laravel pour que les validations soient appliquées correctement. Après cette adaptation, les textareas respectent bien les règles de validation : les champs obligatoires doivent être remplis, et les champs désactivés ne bloquent pas l'envoi du formulaire.

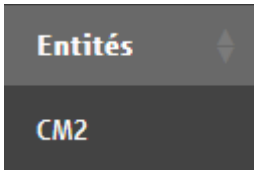
```
return [
    'suen_date_rdv_date' => 'required|date',
    'suen_date_rdv_heure' => 'required|date_format:H:i',
    'suen_motif' => 'required|string|max:350|regex:'.Constantes::REGEX_BASE,
    'suen_garage' => 'nullable|string|max:50|regex:'.Constantes::REGEX_BASE,
    'suen_obs' => 'nullable|string|max:350|regex:'.Constantes::REGEX_BASE,
    'suen_vehi_id' => 'required|exists:vehicules,vehi_id',
```

Liste des règles

Cette correction garantit que toutes les informations saisies dans les textareas sont prises en compte correctement et que le formulaire fonctionne de manière fiable et conforme aux règles définies.

6.9 Ajout de libellés longs et adaptation à la taille d'écran

À l'origine, le tableau affichait uniquement des libellés courts. Cependant, ces libellés manquaient parfois de précision lorsque l'écran était suffisamment large pour afficher davantage d'informations. Il m'a donc été demandé d'intégrer des libellés longs, tout en conservant les libellés courts pour les écrans plus petits.



Exemple de libelle court

J'ai d'abord modifié le code du tableau pour ajouter les libellés longs en plus des libellés courts dans chaque cellule concernée.

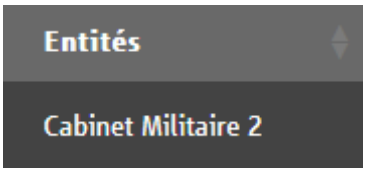
Ensuite, afin d'adapter l'affichage à la taille disponible, j'ai mis en place un script JavaScript. Ce script vérifie la largeur de la cellule du tableau et, en fonction de celle-ci, affiche soit le libellé

long, soit le libellé court. Pour cela, j'ai utilisé la propriété CSS display afin de masquer l'un des deux éléments.

```
<div class="entitesContainerClass">
    @foreach ($entites as $e)
        <div class="entite-line">
            <span class="entite-libelle">{{ $e->enti_libelle }}</span>
            <span class="entite-liblong" style="display: none;">{{ $e->enti_lib_long }}</span>
        </div>
    @endforeach
</div>

<script>
    if (document.querySelector(".entitesContainerClass").offsetWidth > 120) {
        document.querySelectorAll('.entite-liblong').forEach(e => {
            e.style.display = '';
        });
        document.querySelectorAll('.entite-libelle').forEach(e => {
            e.style.display = 'none';
        });
    } else {
        document.querySelectorAll('.entite-liblong').forEach(e => {
            e.style.display = 'none';
        });
        document.querySelectorAll('.entite-libelle').forEach(e => {
            e.style.display = '';
        });
    }
</script>
```

Code HTML et javascript modifié

Lorsque la cellule est suffisamment large, le libellé long est affiché. Dans le cas contraire, le libellé court reste visible.



Exemple de libelle long

6.10 Ajout du grade à côté du nom et du prénom de l'utilisateur

Il m'a été demandé d'ajouter le grade à côté du nom et du prénom de l'utilisateur dans le header de l'application. Cette modification impliquait plusieurs adaptations, aussi bien en base de données que côté back-end et front-end.

Header juste avec nom et prénom

Dans un premier temps, il a fallu modifier la base de données, car cette information n'était pas stockée. Le grade est récupéré depuis le token du SSO, dans les données envoyées lors de la connexion.

```
ALTER TABLE IF EXISTS [redacted]utilisateurs ADD COLUMN UTIL_NOM_AFFICHAGE character varying(255);
```

Requête pour modifier la base de données et ajouter le champ voulu

La donnée à récupérer était nommée display_name. Elle faisait partie du tableau transmis lors de l'authentification SSO. Afin d'éviter d'interroger le SSO à chaque connexion pour récupérer cette information, il a été décidé de l'enregistrer en base de données lors de la première connexion ou lors de la mise à jour des informations utilisateur.

J'ai donc modifié le code PHP afin de récupérer la valeur display_name dans le tableau retourné par le SSO, l'enregistrer dans la base de données, mettre à jour la valeur si nécessaire lors des connexions suivantes.

```php
// UserInfo
$userInfo = $user->user;


   // Extraire les informations nécessaires
   $courriel = $userInfo['email'] ?? null;
   $nom = strtoupper(string: $userInfo['usual_name']) ?? null;
   $prenom = $userInfo['usual_forename'] ?? null;
   $login = $userInfo['preferred_username'] ?? null;
   $mobile = isset($userInfo['mobile'][0]) ? $userInfo['mobile'][0] : null;
$nom_affichage = $userInfo['display_name'] ?? null;
if ($mobile !== null)
    $mobile = '0' . substr(string: $mobile, offset: 3);



   // Appeler la méthode upsertUtilisateur
   $utilisateur = $this->utilisateurService->upsertUtilisateur(
       courriel: $courriel,
       nom: $nom,
       prenom: $prenom,
       login: $login,
       mobile: $mobile,
       nom_affichage: $nom_affichage
   );
```

Code de récupération du display_name

Enfin, il a fallu modifier l'interface du header pour afficher le grade à côté du nom et du prénom de l'utilisateur. L'affichage a été adapté afin d'intégrer cette nouvelle information de manière cohérente avec le design existant.

Header avec le grade ajouté

7. Correction de bugs

7.1 Les retours explicites

Dans le projet, une commande ***make front*** permet d'analyser les fichiers liés au front-end. Cette commande vérifie notamment la présence des docstrings (doc blocks), l'ajout de retours explicites pour chaque fonction, ainsi que la longueur des fonctions.

```
Components/Forms/TextAreaCharacters.php : X X
 __construct() : Ligne 19 : méthode sans Doc Block
 render() : Ligne 30 : méthode sans Doc Block
 render() : Ligne 30 : méthode sans type de retour explicite (ajouter : void si rien n'est retourné)(PSR-12)
View/Components/Immatriculation.php : V V
View/Components/MissionModal.php : V V V
View/Components/ResetButton.php : V V
View/Components/StateButton.php : V V
View/Components/SubmitButton.php : V V
View/Components/VehiculeModal.php : V V

Rapport global
Fichiers analysés : 104
   Bons : 67
   Avec erreurs : 37
   Taux de réussite fichiers : 64.42%

Méthodes analysées : 420
   Correctes : 299
   En erreur : 121
   Taux de réussite méthodes : 71.19%
```

Retour de la commande

Sur certains programmes, les retours explicites n'étaient pas ajoutés ; il m'a donc été demandé de les intégrer. Je n'avais en revanche pas à m'occuper des docstrings ni des autres éléments.

```php
public function render(): View
{
    return view(view: 'components.forms.input-number');
}
```

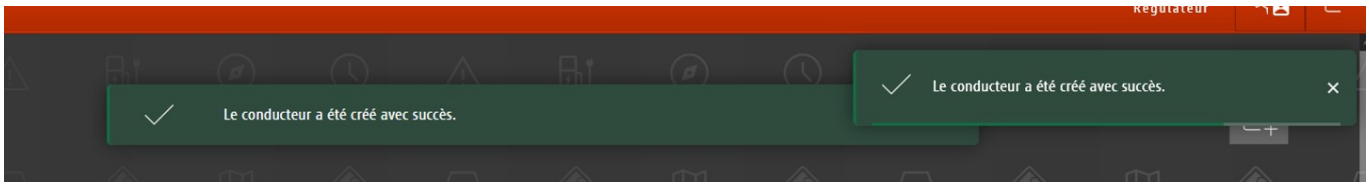
Avant l'ajout des retours explicites

```php
0 references | 0 overrides
public function render(): View
{
    return view(view: 'components.forms.input-number');
}
```

Après l'ajout des retours explicites

7.2 Double pop-up

Quand utilisateur validait la création, la modification ou la suppression d'un élément (par exemple un conducteur), une pop-up devait confirmer que l'élément avait bien été créé, mis à jour ou supprimé. Or, deux pop-ups s'affichaient au lieu d'une seule. Une modification avait été effectuée par l'équipe à un moment donné, sans que l'ancienne implémentation ne soit supprimée. La résolution de ce problème m'a donc été confiée.



Problème de double pop-up

Une analyse du code a permis d'identifier et de supprimer la ligne responsable de l'affichage de la pop-up supplémentaire.

```
@@ -1,4 +1,4 @@
1   -<div id="toast-container" aria-live="polite" aria-atomic="true"></div>
1   +<!-- <div id="toast-container" aria-live="polite" aria-atomic="true"></div> -->
```

Ligne supprimée (en rouge)

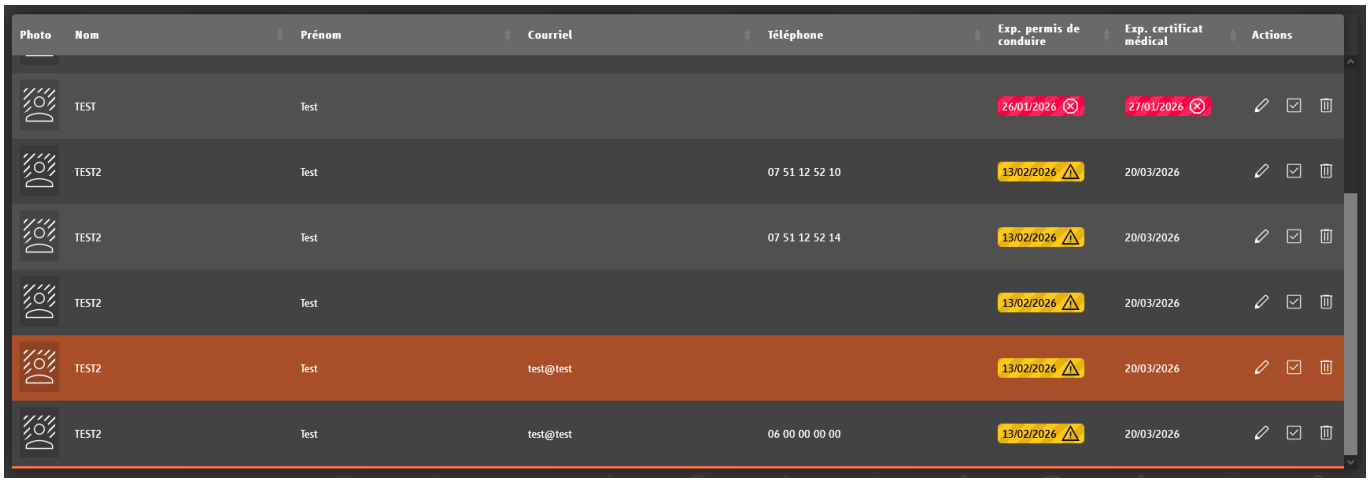
Il a également été nécessaire d'ajouter le composant <x-session-alert /> dans certains fichiers, ceux-ci ne disposant pas de pop-up de confirmation. Ces corrections ont permis de rétablir un fonctionnement cohérent.

```
@extends(view: 'layouts.layout')
@section(section: 'title', content: 'Accueil')
@section(section: 'main')
<h1 class="screen-reader-only">Accueil</h1>
<x-session-alert />
```

Ajout de la ligne demandé dans un fichier

7.3 Erreur conducteur dupliqué à la modification

Lorsqu'un utilisateur modifiait un conducteur, par exemple son numéro de téléphone, une nouvelle ligne était créée au lieu de mettre à jour l'existante.



Exemple de lignes dupliquées

Le formulaire a été vérifié pour s'assurer que la route utilisée pour la modification était correcte, ce qui a été confirmé.

```
<div class="form-container">
    <p>Tous les champs sont obligatoires sauf ceux indiqués comme facultatifs.</p>
    <form class="form" action="{{ route(name: 'conducteurs.modifier.submit') }}" method="post">
        @csrf
        @method('PUT')
        <div class="inputs-area">
            <x-forms.input-hidden name="cond_id" value="{{ $conducteur->cond_id }}" />
            <x-forms.input-text label="Nom" name="cond_nom" placeholder="Renseigner le nom" value='
            <x-forms.input-text label="Prénom" name="cond_prenom" placeholder="Renseigner le prénor
            <x-forms.input-email label="Courriel" name="cond_courriel" placeholder="Renseigner le 
            <x-forms.input-tel label="Téléphone" name="cond_tel" placeholder="Renseigner le téléph
            <x-forms.input-file label="Photo" name="cond_photo" placeholder="Renseigner la photo" 
            <img id="img-upload-conducteur" src="{{ File::exists(path: 'storage/conducteurs/'.$cond
            <x-forms.input-date label="Expiration du permis de conduire" name="cond_permis_date" pl
            <x-forms.input-date label="Expiration du certificat médical" name="cond_med_date" place
        </div>
        <div class="form-buttons-area">
            <x-reset-button />
            <x-submit-button />
        </div>
    </form>
</div>
```

Formulaire de modification

L'analyse a ensuite été poursuivie côté back-end, jusqu'à la fonction associée à cette route. Il a été constaté que la fonction createConducteur était appelée à la place de updateConducteur, définie dans le service conducteur.

```
/**
 * Modifie un conducteur existant.
 *
 * @param ConducteurRequest $requestConducteur La requête validée
 * @return \Illuminate\Http\RedirectResponse
 */
0 references | 0 overrides
public function edit(ConducteurRequest $requestConducteur): \Illuminate\Http\RedirectResponse
{
    $conducteur = $this->conducteurService->findConducteurById(cond_id: $requestConducteur->cond_id);
    return $this->processConducteur(requestConducteur: $requestConducteur, callback: fn($validator): Conducteur => $this->conducteurService->createConducteur(da…$validator));
}
```

createConducteur présent pour la modification

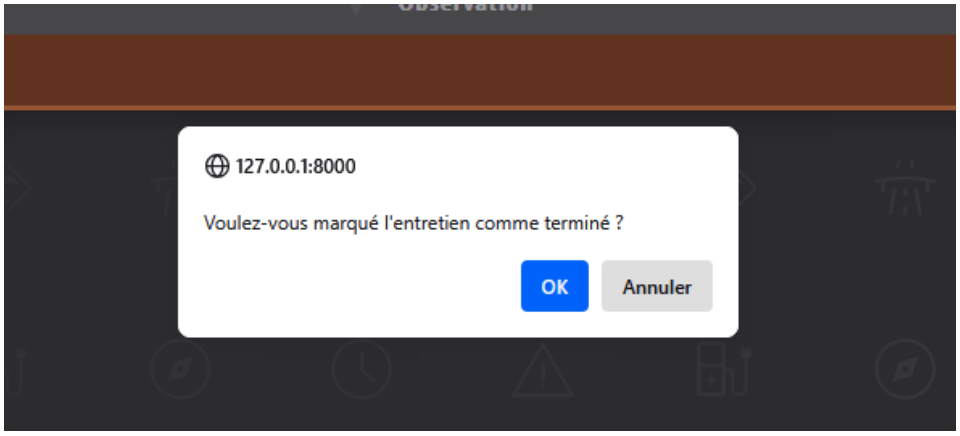
```
/**
 * Modifie un conducteur existant.
 *
 * @param ConducteurRequest $requestConducteur La requête validée
 * @return \Illuminate\Http\RedirectResponse
 */
0 references | 0 overrides
public function edit(ConducteurRequest $requestConducteur): \Illuminate\Http\RedirectResponse
{
    $conducteur = $this->conducteurService->findConducteurById(cond_id: $requestConducteur->cond_id);
    return $this->processConducteur(requestConducteur: $requestConducteur, callback: fn($validator): Conducteur => $this->conducteurService->updateConducteur(co…$conducteur, $validator));
}
```

Programme après modification

La correction de cet appel a permis de résoudre le problème.

7.4 Modification d'une pop-up pour l'entretien de véhicules

Pour valider l'action d'un utilisateur, une pop-up s'affiche. Dans ce cas, il s'agissait d'une pop-up confirmant la fin d'un entretien. Bien que toutes les pop-ups du site aient déjà été modifiées, celle-ci n'avait pas encore été mise à jour.



Pop-up d'origine

Le code associé à cette pop-up a été localisé, puis supprimé et remplacé par un nouveau code adapté.

```
<!-- Terminer ou non un entretien -->
<x-dialog target="TerminerEntretien{{ $e->suen_id }}" text="Marquer l'entretien comme {{ $e->suen_fait ? 'non ' : '' }}terminé" form-action="{{ route(name: 'vehicules.entretiens.state') }}" form-method="PATCH">
    <x-slot name="button">
        <button class="table-button dialog-button" type="submit" data-target="dialogTerminerEntretien{{ $e->suen_id }}">
            <img src="{{ $e->suen_fait ? asset(path: 'img/icons/tab-ligne-activee.svg') : asset(path: 'img/icons/tab-ligne-desactivee.svg') }}" title="Marquer l'entretien comme {{ $e->suen_fait ? 'non ' : '' }}terminé
        </button>
    </x-slot>
    Voulez-vous marquer l'entretien comme {{ $e->suen_fait ? 'non ' : '' }}terminé ?
    <x-slot name="inputs">
        <input type="hidden" name="suen_id" value="{{ $e->suen_id }}">
    </x-slot>
</x-dialog>
```

Programme pour la nouvelle pop-up

La pop-up a ainsi été modifiée avec succès, ce qui nous donne :

```
<x-suivi-entretien-modal :entr
<form action="{{ route(name: '
    @csrf
    @method('PATCH')
    <input type="hidden" name=
    <x-state-button :state="$e
</form>
```

Programme lié à

Pop-up après modifications

7.5 Réouverture de l'overlay après la validation du formulaire

Pour un overlay qui se rouvrait après la validation du formulaire, un script JavaScript a été ajouté au bouton de validation, lequel est censé fermer le formulaire / l'overlay. Ce script utilise un écouteur d'événement permettant de fermer l'overlay lors du clic sur le bouton.

```
document.querySelectorAll(".green-button").forEach(button => {
    const modal = button.closest('.modal-overlay');
    if (modal) {
        const target = `#${modal.id}`;
        button.addEventListener("click", () => closeModal(target));
    }
});
```

Script pour fermer l'overlay au clique sur le bouton de validation

7.6 Majuscule non appliquée à la colonne ‘nom’ lors de la création d’un utilisateur

Lors de la création d’un utilisateur, la valeur renseignée dans la colonne « nom » n’était pas automatiquement mise en majuscule, ce qui entraînait des problèmes d’affichage.

	util_id [PK] bigint	util_nom character varying (255)	util_prenom character varying (255)	util_login character varying (255)	util_courriel character varying (255)	util_role_id integer	created_at timestamp without time zone	updated_at timestamp without time zone
102	11	Dumont	Nicolas	n.dumont	nicolas.dumont@intradef.gouf.fr	4	2026-02-03 09:18:26	2026-02-03 09:18:26

Problème initial : nom non affiché en majuscules

Le problème a été corrigé par la mise en place d’un trigger exécuté before insert (avant l’ajout d’un enregistrement dans la base). Celui-ci met automatiquement le nom en majuscules lors de l’ajout d’un nouvel enregistrement en base de données.

```
CREATE OR REPLACE FUNCTION upper_nom_trigger()
RETURNS TRIGGER AS $$
BEGIN
    NEW.util_nom = UPPER(NEW.util_nom);
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER upper_nom
BEFORE INSERT ON [redacted].utilisateurs
FOR EACH ROW
EXECUTE FUNCTION upper_nom_trigger();
```

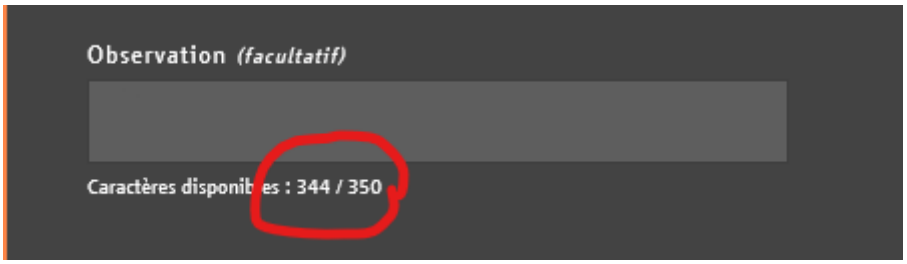
Trigger créé pour mettre automatiquement le nom en majuscules

	util_id [PK] bigint	util_nom character varying (255)	util_prenom character varying (255)	util_login character varying (255)	util_courriel character varying (255)	util_role_id integer	created_at timestamp without time zone	updated_at timestamp without time zone
103	119	DUMONT	Nicolas	n.dumont	nicolas.dumont@intradef.gouf.fr	4	2026-02-03 09:51:21	2026-02-03 09:51:21
104	120	ZERON	Nina	n.zeron	nina.zeron@intradef.gouf.fr	4	2026-02-03 09:51:42	2026-02-03 09:51:42
105	121	THOMAS	Yves	y.thomas	yves.thomas@intradef.gouf.fr	4	2026-02-03 09:52:11	2026-02-03 09:52:11

Données mis en majuscules

7.7 Réinitialisation des caractères disponibles

Lors de l’annulation de la modification du formulaire, le compteur indiquant le nombre de caractères restants sous les champs de texte ne se réinitialisait pas correctement.



Nombre de caractère non réinitialisé

Le problème a été corrigé en ajoutant un code qui recalcule et affiche le nombre de caractères disponibles à chaque clic sur le bouton Reset, assurant que le compteur affiche toujours la valeur correcte.

```
function initCharCounters() {
    const textareas = document.querySelectorAll("textarea");
    textareas.forEach((textarea) => {
        let max_length = textarea.getAttribute("maxlength") || 100;
        max_length = parseInt(max_length);
        textarea.setAttribute("maxlength", max_length);
        // Trouver le span suivant associé
        const charDiv = textarea.nextElementSibling;
        if (charDiv && charDiv.classList.contains("char-length")) {
            // Mise à jour initiale avec le texte déjà présent
            const initial_length = textarea.value.length;
            charDiv.innerText = `Caractères disponibles : ${max_length - initial_length} / ${max_length}`;
        }
        textarea.addEventListener("input", function () {
            let value_length = this.value.length;
            if (value_length > max_length) {
                this.value = this.value.substring(0, max_length);
                value_length = this.value.length;
            }
            if (charDiv) {
                charDiv.innerText = `Caractères disponibles : ${max_length - value_length} / ${max_length}`;
            }
        });
    });
}

document.addEventListener("DOMContentLoaded", initCharCounters);

// Ecouteur au reset d'un formulaire réinitialiser aussi le nombre de caractères disponibles
document.querySelectorAll("form").forEach(el => {
    el.addEventListener("reset", function() {
        setTimeout(() => {
            initCharCounters();
        }, 10);
    });
});

```

Programme de réinitialisation des caractères

8. Fonctionnalités ajoutées

8.1 Système d'export PDF et Excel

L'application disposait déjà d'un système d'export au format PDF et Excel, qui devait être remplacé en raison de certains problèmes. Les bibliothèques laravel-dompdf (pour les fichiers PDF) et laravel-excel (pour les fichiers Excel) ont été intégrées à l'application afin de proposer une solution plus stable.

Une fois ces bibliothèques installées, elles ont été utilisées pour générer les fichiers PDF. Les fichiers Livewire de Laravel, utilisés pour les boutons d'export, ont été appelés afin de récupérer les données au moment du clic.

```
<button wire:click="generatePdf">Export PDF</button>
<button wire:click="generateExcel">Export Excel</button>
```

Utilisation des fonctions du livewire pour la création des boutons

```
/**
 * Envoie les données vers la Class Export pour générer un PDF
 * @return StreamedResponse
 */
0 references | 0 overrides
public function generatePdf(): StreamedResponse
{
    $data=[
        'datas' => (new MissionService())->utilisateur(util_id: Auth::user()->util_id, perPage: $this->perPage, sortField: $this->sortField, sortDirection: $this->sortDirection, unfinished: $this->unfinished),
        'title' => 'Liste des missions',
        'colonnes' => ['Référence', 'État', 'Début', 'Type', 'Bénéficiaire', 'Passagers'],
        'links' => ['id', 'dernierEtatMission->etatMission->etmi_libelle', 'oldestTrajet->traj_debut', 'typeMission->tymi_libelle|$mission->typeMission->miss_esco', 'benef_dema_name', 'miss_nb_pass'],
        'design' => ['bold', '', '', '', '', ''],
        'format' => 'landscape', // landscape pour format paysage, portrait ou ne rien mettre pour format portrait
        'nom_fichier' => 'mission'
    ];

    return Export::toPdf(data: $data);
}
```

Fonction utilisée au clic de l'utilisateur pour générer un PDF

```
/**
 * Envoie les données vers la Class Export pour générer un fichier Excel
 * @return BinaryFileResponse
 */
0 references | 0 overrides
public function generateExcel(): BinaryFileResponse
{
    $data=[
        'datas' => (new MissionService())->utilisateur(util_id: Auth::user()->util_id, perPage: $this->perPage, sortField: $this->sortField, sortDirection: $this->sortDirection, unfinished: $this->unfinished),
        'colonnes' => ['Référence', 'État', 'Début', 'Type', 'Bénéficiaire', 'Passagers'],
        'links' => ['id', 'dernierEtatMission->etatMission->etmi_libelle', 'oldestTrajet->traj_debut', 'typeMission->tymi_libelle|$mission->typeMission->miss_esco', 'benef_dema_name', 'miss_nb_pass'],
        'nom_fichier' => 'mission'
    ];

    return Export::toExcel(data: $data);
}
```

Fonction utilisée au clic de l'utilisateur pour générer un fichier Excel

Une classe Export a été créée avec une fonction toPdf, permettant de transformer les données envoyées par le bouton en fichier PDF téléchargeable par l'utilisateur.

```php
/**
 * Fonction pour télécharger un tableau au format PDF
 * @param array $data
 * @return StreamedResponse
 */
1 reference | 0 overrides
public static function toPdf(array $data): StreamedResponse
{
    $pdf=Pdf::loadView(view: 'components.export.export-pdf', data: $data);

    return response()->streamDownload(callback: function() use($pdf): void{
        echo $pdf->stream();
    }, name: $data['nom_fichier'].'.pdf');
}
```

Fonction toPdf

Le contenu du PDF est basé sur un fichier Blade (template Laravel), codé pour s'adapter à différents types de tableaux. Une fois généré, le PDF dispose d'un premier design qui pourra être ajusté ultérieurement selon les demandes des utilisateurs.

Liste des missions

Référence	État	Début	Type	Bénéficiaire	Passagers
EV2600010	Demande approuvée	2026-01-07 08:45:00	VL	UTILISATEUR Monsieur	1
EX2600009	Prestation terminée	2026-01-07 08:45:00	VELO	UTILISATEUR Monsieur	1
RT2500001	Demande rejetée	2025-11-14 10:15:00	TAXI	UTILISATEUR Monsieur	5
RT2600007	Demande initiée	2026-01-07 08:45:00	TAXI	UTILISATEUR Monsieur	1

Fichier PDF généré

Le même principe a été appliqué pour les fichiers Excel : une fonction toExcel récupère les données au clic sur le bouton et génère un fichier Excel.

```php
/**
 * Fonction pour télécharger un tableau au format xlsx (Excel)
 * @param array $data
 * @return void
 */
1 reference | 0 overrides
public static function toExcel(array $data): BinaryFileResponse
{
    // Nom du fichier par défaut si non spécifié
    $filename = $data['nom_fichier'] ?? 'export';

    // Créer une classe anonyme pour l'export
    $export = new class(data: $data) implements FromView {
        protected $data;

        public function __construct(array $data)
        {
            $this->data = $data;
        }

        public function view(): View
        {
            return view(view: 'components.export.export-excel', data: $this->data);
        }
    };

    // Exporter le fichier Excel
    return Excel::download(export: $export, fileName: $filename . '.xlsx');
}
```

Fonction toExcel

Contrairement aux PDF, Laravel-Excel ne transforme pas directement un fichier HTML en Excel. Il utilise généralement des collections ou des tableaux PHP pour construire le fichier. Cependant, le rendu a été adapté pour correspondre à la mise en forme souhaitée, en s'inspirant du template HTML utilisé pour les PDF.

Le résultat final permet à l'utilisateur de télécharger des fichiers PDF et Excel correctement formatés et facilement exploitables.

	A	B	C	D	E	F
1	Référence	État	Début	Type	Bénéficiaire	Passagers
2	EV2600010	Demande approuvée	2026-01-07 08:45:00	VL	UTILISATEUR Monsieur	1
3	EX2600009	Prestation terminée	2026-01-07 08:45:00	VELO	UTILISATEUR Monsieur	1
4	RT2500001	Demande rejetée	2025-11-14 10:15:00	TAXI	UTILISATEUR Monsieur	5
5	RT2600007	Demande initiée	2026-01-07 08:45:00	TAXI	UTILISATEUR Monsieur	1

Fichier Excel généré

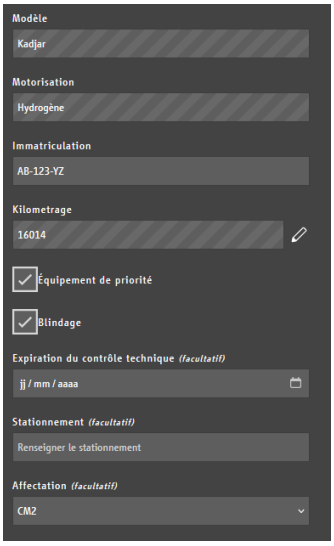
8.2 Ajout du suivi de kilométrage dans le formulaire de modification d'un véhicule

Dans le formulaire de modification d'un véhicule, il m'a été demandé d'ajouter l'affichage du kilométrage actuel, ainsi qu'un bouton permettant d'ouvrir une pop-up pour enregistrer un nouveau kilométrage. Cette fonctionnalité a été mise en place temporairement.

J'ai d'abord modifié le HTML du formulaire existant afin d'y intégrer l'affichage du kilométrage ainsi qu'un bouton dédié à l'ajout d'une nouvelle valeur.

```
<div class="input-area-modif">
    <div class="f-align-column">
        <x-forms.input-text label="Kilometrage" name="suivi_kilo" placeholder="Kilométrage du véhicule" :value="$lastKilos['suki_kilo_nve']" required disabled/>
    </div>
    <button type="button" class="modal-trigger table-button modal-button" title="Ajouter" aria-label="Ajouter" data-target="#suiviKilo{{ $vehicule->vehi_id }}">
        <img src="{{ asset(path: 'img/icons/tab-modifier.svg') }}" title="Ajouter" alt="Ajouter">
    </button>
</div>
```

HTML du formulaire modifié

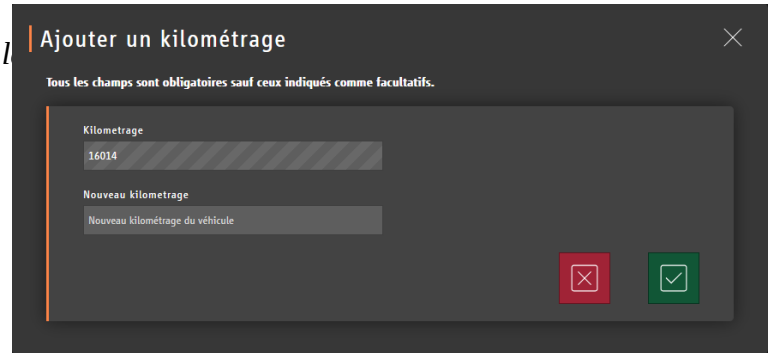


Formulaire modifié

J'ai ensuite créé le HTML de la modale, qui s'ouvre lors du clic sur le bouton. Cette modale contient un champ permettant de saisir le nouveau kilométrage du véhicule.

```
@props([
    'kilometrages' => null,
    'vehicule', // Pour unique ID si pas de suen_id
    'lastKilos' => null,
])

<div id="suiviKilo{{ $vehicule }}" class="modal-overlay" style="display: none;">

    <div class="modal-content" role="dialog" aria-modal="true" aria-labelledby="modal-title-{{ $vehicule }}">
        <div class="modal-header">
            <span id="modal-title-{{ $vehicule }}">Ajouter un kilométrage</span>
            <button class="modal-close" data-modal-close type="button" aria-label="Fermer">
                <img src="{{ asset(path: 'img/icons/fermer.svg') }}" alt="Fermer" style="width: 42px;">
            </button>
        </div>

        <div class="modal-body">

            <div class="form-container">
                <p>Tous les champs sont obligatoires sauf ceux indiqués comme facultatifs.</p>
                <form class="form formulaire" action="{{ route(name: 'suivikilo.store') }}" method="POST">
                    @csrf
                    @method('POST')
                    <input type="hidden" name="suki_vehi_id" value="{{ $vehicule }}">
                    <input type="hidden" name="suki_kilo_anc" value="{{ $lastKilos['suki_kilo_nve'] }}">
                    <div class="inputs-area">
                        <x-forms.input-text label="Kilometrage" name="ancien_kilometrage" placeholder="Kilométrage du véhicule" :value="$lastKilos['suki_kilo_nve'] ?? 'Aucun kilométrag
                        <x-forms.input-text label="Nouveau kilometrage" name="suki_kilo_nve" placeholder="Nouveau kilométrage du véhicule" required/>
                    </div>
                    <div class="form-buttons-area">
                        <x-reset-button />
                        <x-submit-button />
                    </div>
                </form>
            </div>
        </div>
    </div>
</div>
```

Programme lié à l



Le rendu de la modale

Côté back-end, j'ai ajouté une fonction store afin d'enregistrer les nouveaux kilométrages en base de données. Cette approche permet de créer une nouvelle entrée liée au véhicule concerné, ce qui ouvre la possibilité de conserver un historique plutôt que d'écraser la valeur existante.

```
+     public function store(SuiviKilometrageRequest $request): RedirectResponse {
+         // Valider les champs du formulaire
+         $validator = $request->all();
+
+         DB::beginTransaction();
+         try {
+             // Création d'un suivi d'entretien
+             $createdRecord = SuiviKilometrage::create($validator);
+
+             DB::commit();
+             // Redirection vers la modification d'un véhicule
+             return redirect()->route('vehicules.modifier', base64_encode($createdRecord->suki_vehi_id))->with('success', "Le
suivi de kilométrage a été créé avec succès.");
+
+         } catch (Exception $e) {
+             DB::rollBack();
+             return redirect()->back()->withInput()->with('fail', "Échec d'enregistrement, problème temporaire.");
+         }
+     }
```

Fonction store ajouté

Afin d'assurer la validité des données saisies, j'ai également créé un SuiviKilometrageRequest dans Laravel. Ce fichier contient les règles de validation spécifiques au kilométrage, garantissant que les valeurs enregistrées respectent les contraintes définies.

```
+ class SuiviKilometrageRequest extends FormRequest
+ {
+     /**
+      * Détermine si l'utilisateur est autorisé à faire cette requête.
+      *
+      * @return bool Vrai si l'utilisateur est autorisé
+      */
+     public function authorize(): bool
+     {
+         return $this->routeIs('suivikilo.*');
+     }
+
+     /**
+      * Récupère les règles de validation qui s'appliquent à la requête.
+      *
+      * @return array Règles de validation
+      */
+     public function rules(): array
+     {
+         $rules = $this->getBaseRules();
+
+         return $rules;
+     }
+
+     /**
+      * Obtient les règles de validation de base.
+      *
+      * @return array Règles de validation
+      */
+     protected function getBaseRules(): array
+     {
+         return [
+             'suki_kilo_anc' => 'required|numeric',
+             'suki_kilo_nve' => 'required|numeric|gt:suki_kilo_anc',
+             'suki_vehi_id' => 'required|numeric',
+             'suki_miss_id' => 'numeric',
+         ];
+     }
+ }
```

Code de la request

Ainsi, l'ajout d'un nouveau kilométrage peut désormais se faire directement depuis le formulaire de modification du véhicule via la modale dédiée. La valeur est correctement enregistrée et

prise en compte par l'application, ce qui permet un fonctionnement fluide et conforme à la demande initiale, même si cette solution reste susceptible d'évoluer ou d'être retirée par la suite.

9. Gestion des données

9.1 Backup automatisé des données

Dans le cadre du projet, j'ai été chargé de mettre en place un système de backup automatisé en particulier la table testing, utilisée pour les tests. L'objectif était de pouvoir restaurer la base rapidement en cas de problème.

J'ai commencé par chercher la commande adaptée pour effectuer un backup de cette table. Après avoir identifié la syntaxe correcte, j'ai testé la commande manuellement afin de vérifier que le fichier généré contenait bien toutes les données nécessaires et pouvait être utilisé pour une restauration ultérieure.

```
C:\dumps>cd /d C:\dumps && "C:\Program Files\PostgreSQL\15\bin\pg_dump.exe" -h 10.32.245.130 -p 5432 -U postgres -d test
ing -n [redacted] -Fp -f "C:\dumps\dump_%DATE:~-4,4%%DATE:~-10,2%%DATE:~-7,2%_%TIME:~0,2%%TIME:~3,2%%TIME:~6,2%.backup"
```

Commande de dump pour la base de données

Une fois cette étape validée, j'ai automatisé la tâche en utilisant les Schedulers Windows. Cela permet de lancer le backup à intervalles réguliers, sans intervention manuelle. J'ai configuré les paramètres pour que chaque fichier soit daté et stocké dans un répertoire donné, garantissant ainsi un historique clair des sauvegardes.

Name	Status	Triggers	Next Run Time	Last Run Time	Last Run Result	Author	Created
Dump journalier te...	Ready	At 8:00 PM on 2/5/2026 - After triggered, repeat every 1.00:00:00 indefinitely.	2/5/2026 8:00:00 PM	2/5/2026 1:34:26 PM	The operation completed succes...	PICSEL\m.rocher.ext	2/5/2026 10:56:14 AM
Dump journalier dev	Ready	At 8:00 PM on 2/5/2026 - After triggered, repeat every 1.00:00:00 indefinitely.	2/5/2026 8:00:00 PM	11/30/1999 12:00:00 AM	The task has not yet run. (0x41303)	PICSEL\m.rocher.ext	2/5/2026 10:56:14 AM

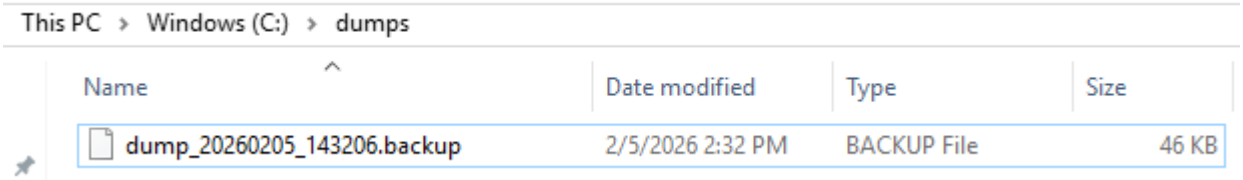
General | Triggers | Actions | Conditions | Settings | History

When you create a task, you must specify the action that will occur when your task starts. To change these actions, open the task property pages using the Properties command.

Action	Details
Start a program	cmd.exe /C "cd /d C:\dumps && "C:\Program Files\PostgreSQL\15\bin\pg_dump.exe" -h 10.32.245.130 -p 5432 -U postgres -d testing -n [redacted] -Fp -f "C:\dumps\dump_%DATE:~-4,4%%DATE:~-10,2%%DATE:~-7,2%_%TIME:~0,2%%TIME:~3,2%...

Scheduler mis en place

Après plusieurs tests, j'ai pu confirmer que les backups s'exécutaient correctement et que la table testing pouvait être restaurée en cas de besoin. Cette mise en place assure la sécurité et la disponibilité des données de test, et facilite la maintenance de l'application lors des développements et essais.



Vue du fichier de backup