

08 septembre

EPCF1

2023

---

Dossier projet – Plateforme WEB pour la gestion des tickets  
(Plateforme Ticketing)

EXPERNET

## Table des matières

I.	Liste des compétences du référentiel qui sont couvertes par le projet	2
II.	Résumé du projet en anglais d'une longueur d'environ 5 à 10 lignes soit 100 à 120 mots, ou environ 600 caractères espaces non compris	3
III.	Cahier des charges, expression des besoins, ou spécifications fonctionnelles du projet	4
IV.	Spécifications techniques du projet, élaborées par le candidat, y compris pour la sécurité	6
V.	Réalisations du candidat comportant les extraits de code les plus significatifs, et en les argumentant, y compris pour la sécurité	8
	Modèle conceptuel des données	8
	Modèle physique des données	9
	Conception et développement de la plateforme	10
	Connexion à la plateforme	11
	Conception	11
	Développement	12
	Page de connexion finale	13
	Dashboard pour les administrateurs	14
	Conception	14
	Développement	15
VI.	Présentation du jeu d'essai élaboré par le candidat de la fonctionnalité la plus représentative (données en entrée, données attendues, données obtenues)	23
VII.	Description de la veille, effectuée par le candidat durant le projet, sur les vulnérabilités de sécurité	29
VIII.	Description d'une situation de travail ayant nécessité une recherche, effectuée par le candidat durant le projet	30

## I. Liste des compétences du référentiel qui sont couvertes par le projet

<b>Concevoir et développer des composants d'interface utilisateur en intégrant les recommandations de sécurité</b>	Développer la partie back-end d'une interface utilisateur web
	Développer des composants d'accès aux données
	Développer une interface utilisateur de type desktop
	Développer la partie front-end d'une interface utilisateur web
	Maquetter une application

Le projet couvre obligatoirement les compétences suivantes :

- Maquetter une application
- Développer des composants d'accès aux données
- Développer la partie front-end d'une interface utilisateur web
- Développer la partie back-end d'une interface utilisateur web

## II. Résumé du projet en anglais d'une longueur d'environ 5 à 10 lignes soit 100 à 120 mots, ou environ 600 caractères espaces non compris

A company requires an online ticket management platform that enables various departments to create tickets for tasks and monitor their progress.

Each ticket can encompass multiple processes, each with its own status, effective dates, and the user who treat.

Users are divided into two groups: 'regular' users and administrators. Administrators have access to a dashboard that compiles statistics on pending, in-progress, and closed tickets per department, as well as user account management. 'Regular' users can view and create tickets, take ownership of ongoing tickets, and close those they are currently working on.

The platform is secured through an authentication system and developed in PHP using the Symfony framework for the back-end, along with daisyUI and Tailwind CSS utilities for the front-end.

### III. Cahier des charges, expression des besoins, ou spécifications fonctionnelles du projet

Le présent cahier des charges définit les exigences et les spécifications pour le développement d'une plateforme de gestion des tickets en ligne pour une entreprise.

- Contexte

Une entreprise souhaite mettre en place une solution permettant à ses différents services de créer des tickets pour effectuer des tâches et de suivre leur progression.

- Fonctionnalités

- Les fonctionnalités permettent de savoir ce qui est fonctionnel pour utilisateur 'régulier' et un administrateur

1. La plateforme a une page de connexion

- Identification de l'utilisateur connecté

- La plateforme identifie l'utilisateur et permet de savoir le type d'utilisateur (régulier ou administrateur).
- L'utilisateur peut se déconnecter.
- L'utilisateur peut demander à réinitialiser son mot de passe depuis la plateforme (il doit recevoir un mail contenant son nouveau mot de passe fort qui est généré aléatoirement).

2. Fonctionnalités utilisateur 'régulier'

- Création de ticket

- L'utilisateur peut créer un ticket avec un raison de création et le service concerné.

- Suivi de ticket

- Chaque ticket peut contenir plusieurs processus, chacun avec son propre statut, des dates effectives, et un utilisateur en charge.

- Prise en charge de ticket

- L'utilisateur peut ouvrir un traitement pour un ticket de son service qui a le statut 'EN ATTENTE'.
- L'utilisateur peut prendre le relais sur le traitement d'un ticket de son service qui a le statut 'EN COURS'.
- L'utilisateur peut clore son traitement sur le ticket et ainsi clore le ticket ce qui marque le statut comme 'Fermé'.
- L'utilisateur peut transférer le ticket à un autre service ce qui marque le statut comme 'TRANSFÉRÉ // EN ATTENTE'.

### 3. Fonctionnalités administrateur

#### **L'administrateur doit avoir les mêmes fonctionnalités que l'utilisateur.**

- Tableau de bord
  - L'administrateur doit avoir accès depuis un tableau de bord aux statistiques des états de tickets par service (nombre de tickets 'EN ATTENTE', nombre de tickets 'EN COURS', nombre de tickets 'Fermé').
  - L'administrateur doit pouvoir gérer les comptes des autres utilisateurs (création d'utilisateur, modification d'un utilisateur, suppression d'un utilisateur).
  - L'administrateur doit avoir accès aux statistiques de chaque utilisateurs (nombre de tickets 'Crée', nombre de ticket 'Pris en charge', nombre de ticket 'Fermé').
- Technologies utilisées
  - Conception de la plateforme
    - Modèle conceptuel de données avec Looping.
    - Diagramme d'utilisation avec Draw.io.
    - Maquettes de la plateforme avec Figma.
  - Développement de la plateforme
    - Back-end en PHP 8.2.4
    - Front-end en CSS 3
  - Persistance des données
    - La base de données avec MySQL 8.1.0 (serveur local avec WampServer).
- Livrable et délais
  - La plateforme complète et fonctionnelle doit être livrée conformément aux spécifications.
  - La date de livraison prévue pour la plateforme est le 08/09/2023.

## IV. Spécifications techniques du projet, élaborées par le candidat, y compris pour la sécurité

### 1. Développement de la plateforme

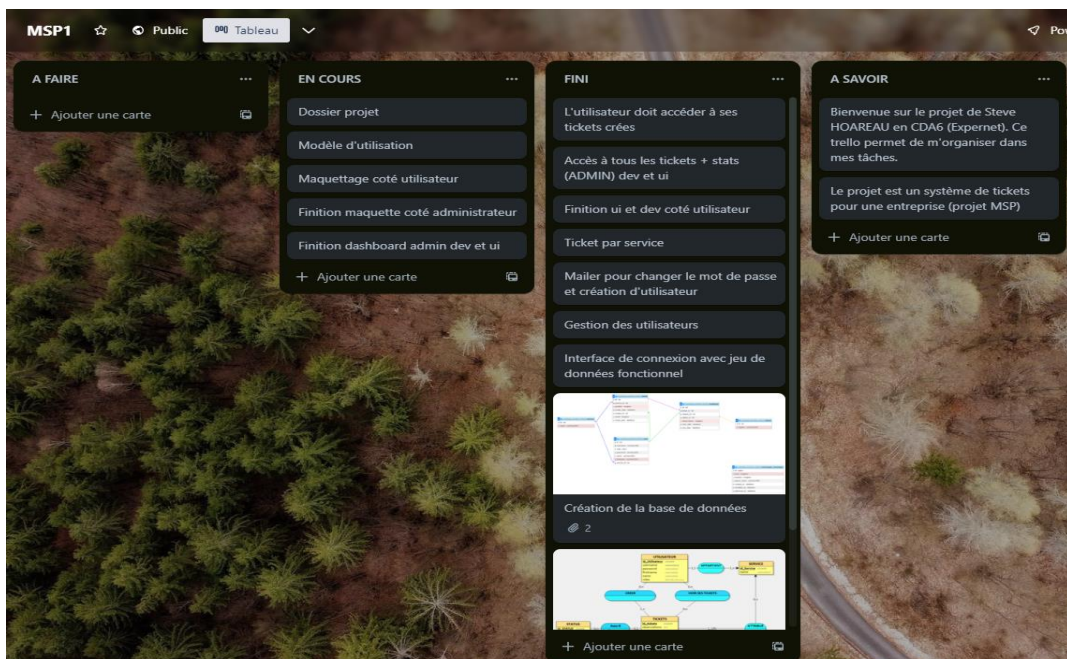
- Le back-end a été codé en PHP avec le Framework Symfony 6.3.1.
- Le front-end a été codé en CSS 3 avec la librairie daisyUI 3.6.4 et l'utilitaire TailwindCSS 3.3.3.
- La gestion des composants de développement s'est faite avec Composer 2.6.2 et NodeJS 18.16.0.
- Les ressources (assets) utilisées sont traitées par Webpack et PostCSS.
- Outil de version avec Git 2.42.0.
- Requêtes AJAX avec JQuery 3.7.1.
- Quelques rendus graphiques avec ChartJS 4.3.2.

### 2. Sécurité de la plateforme

- Un système d'identification est mis en place sur la plateforme avec une page de connexion.
- Les mots de passes sont générés automatiquement depuis l'application et ils sont forts (mot de passe long et contenant des lettres majuscules et minuscules, des symboles et des chiffres).
- Les mots de passes des utilisateurs sont cryptés depuis la plateforme en utilisant bCrypt.
- Les routes dans le code sont sécurisées en vérifiant si l'utilisateur est connecté.
- La manipulation de la base de données dans le code est sécurisée avec des jetons d'accès.
- Les classes dans le code sont encapsulées.

### 3. Gestion de projet

- Gestion des tâches avec Trello ([MSP1 | Trello](#))

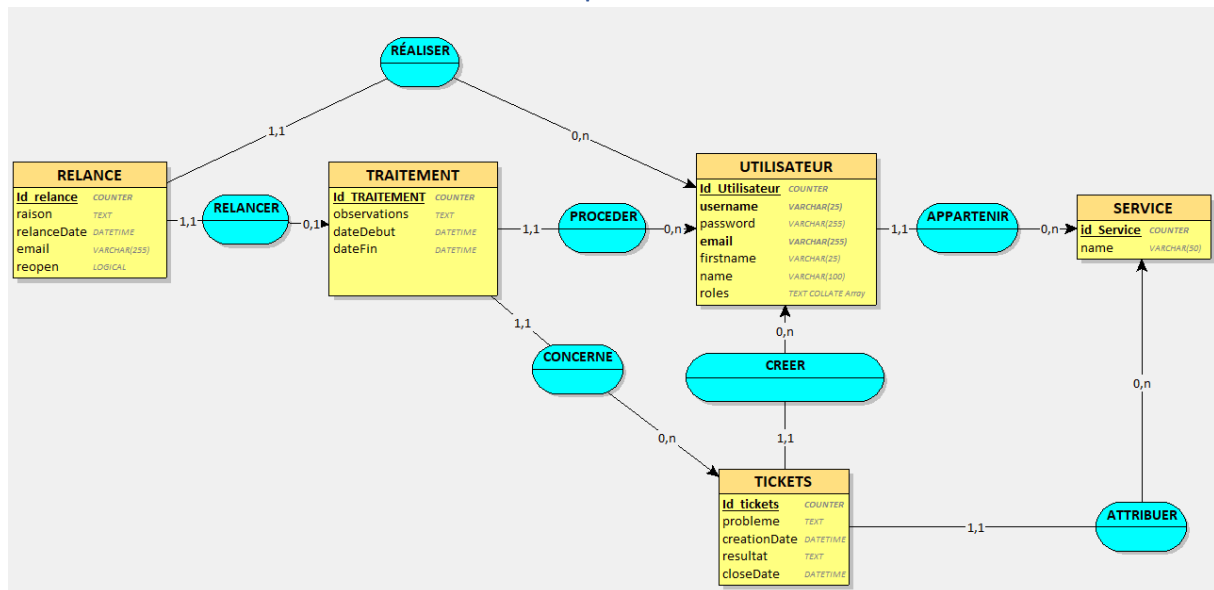


7



## V. Réalisations du candidat comportant les extraits de code les plus significatifs, et en les argumentant, y compris pour la sécurité

Modèle conceptuel des données



Dans ce MCD, on retrouve 5 entités et 6 associations.

L'entité « utilisateur » possède un identifiant « id\_utilisateur ».

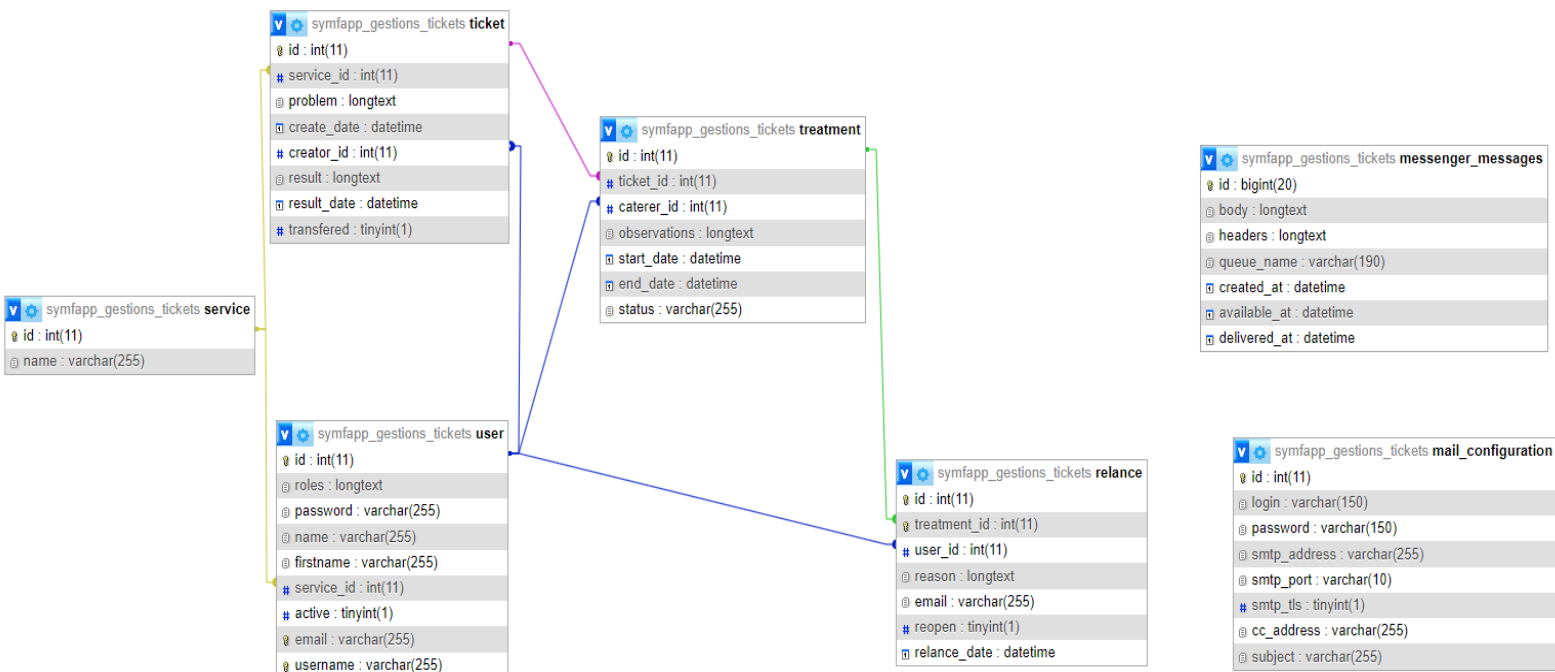
L'entité « service » possède un identifiant « id\_service » et est associé à 0 ou plusieurs utilisateurs, l'utilisateur appartient à 1 et 1 seul service.

L'entité « tickets » possède un identifiant « id tickets » et est créer par à un et un seul utilisateur, l'utilisateur peut créer de 0 à n tickets. Un ticket est aussi attribué à 1 et 1 seul service et un service peut avoir 0 à n tickets.

L'entité « traitement » possède un identifiant « id\_traitement » et est associé à un et un seul utilisateur, l'utilisateur peut procéder 0 à n traitement. Un traitement est aussi concerné par 1 et 1 seul ticket et un ticket peut être concerné par 0 à n traitements.

L'entité « relance » possède un identifiant « id\_relance » et est relancer sur un 1 et 1 seul traitement et un traitement peut avoir une relance. Une relance est réalisée par 1 et 1 seul utilisateur et un utilisateur peut réaliser 0 à n relance.

## Modèle physique des données



On observe 7 tables.

Les 5 entités du modèle conceptuel des données se sont transformées en tables dans le modèle physique. Aucune association a été transformée en table.

On retrouve les anciennes entités :

- « service » qui est devenue la table « service » qui possède une clé primaire « id »
- « utilisateur » qui est devenue la table « user » qui possède une clé primaire « id » et une clé étrangère « service\_id » qui fait référence à la table « service »
- « tickets » qui est devenue la table « ticket » qui possède une clé primaire « id », une clé étrangère « service\_id » qui fait référence à la table « service » et une clé étrangère « creator\_id » qui fait référence à la table « user »
- « traitement » qui est devenue la table « treatment » qui possède une clé primaire « id », une clé étrangère « ticket\_id » qui fait référence à la table « ticket » et une clé étrangère « caterer\_id » qui fait référence à la table « user »
- « relance » qui est devenue la table « relance » qui possède une clé primaire « id », une clé étrangère « treatment\_id » qui fait référence à la table « treatment » et une clé étrangère « user\_id » qui fait référence à la table « user »

On retrouve aussi 2 nouvelles tables :

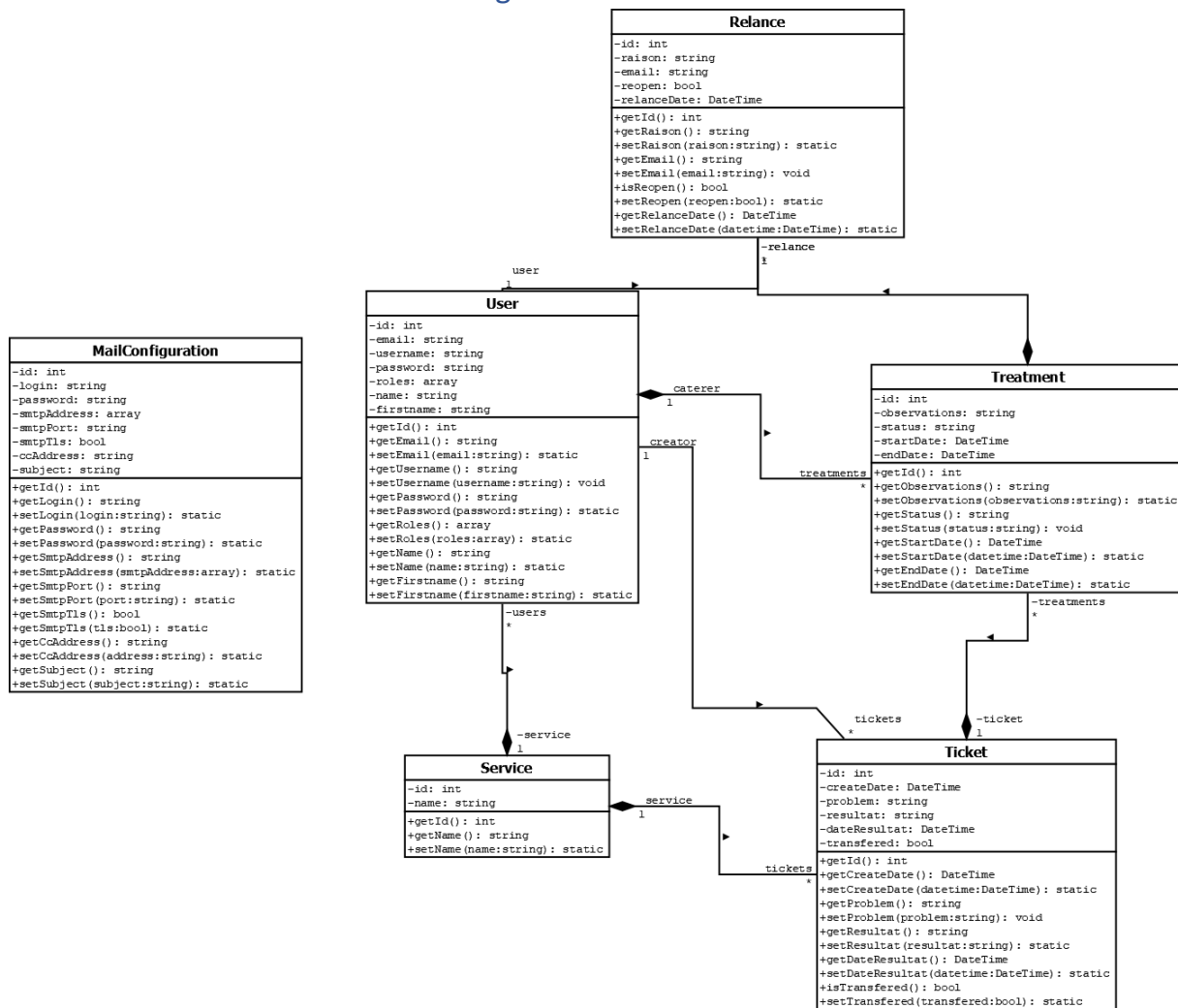
- « messenger\_message » qui possède une clé primaire. Cette table va servir à la plateforme pour enregistrer les messages des Publishers pour être traités avec un système de queue.

```
###> symfony/messenger ###
# Choose one of the transports below
# MESSENGER_TRANSPORT_DSN=amqp://guest:guest@localhost:5672/%2f/messages
# MESSENGER_TRANSPORT_DSN=redis://localhost:6379/messages
MESSENGER_TRANSPORT_DSN=doctrine://default?auto_setup=0
###< symfony/messenger ###
```

En l'occurrence, dans mon application, ce module est configuré dans mon fichier d'environnement «.env».

- « mail\_configuration » qui possède une clé primaire. Cette table va servir à la configuration mail depuis la plateforme et enregistrer celle-ci dans la base de données. Il existera une seule configuration mail dans le cas de l'utilisation de cette plateforme mais ça peut être utile si la plateforme doit évoluer.

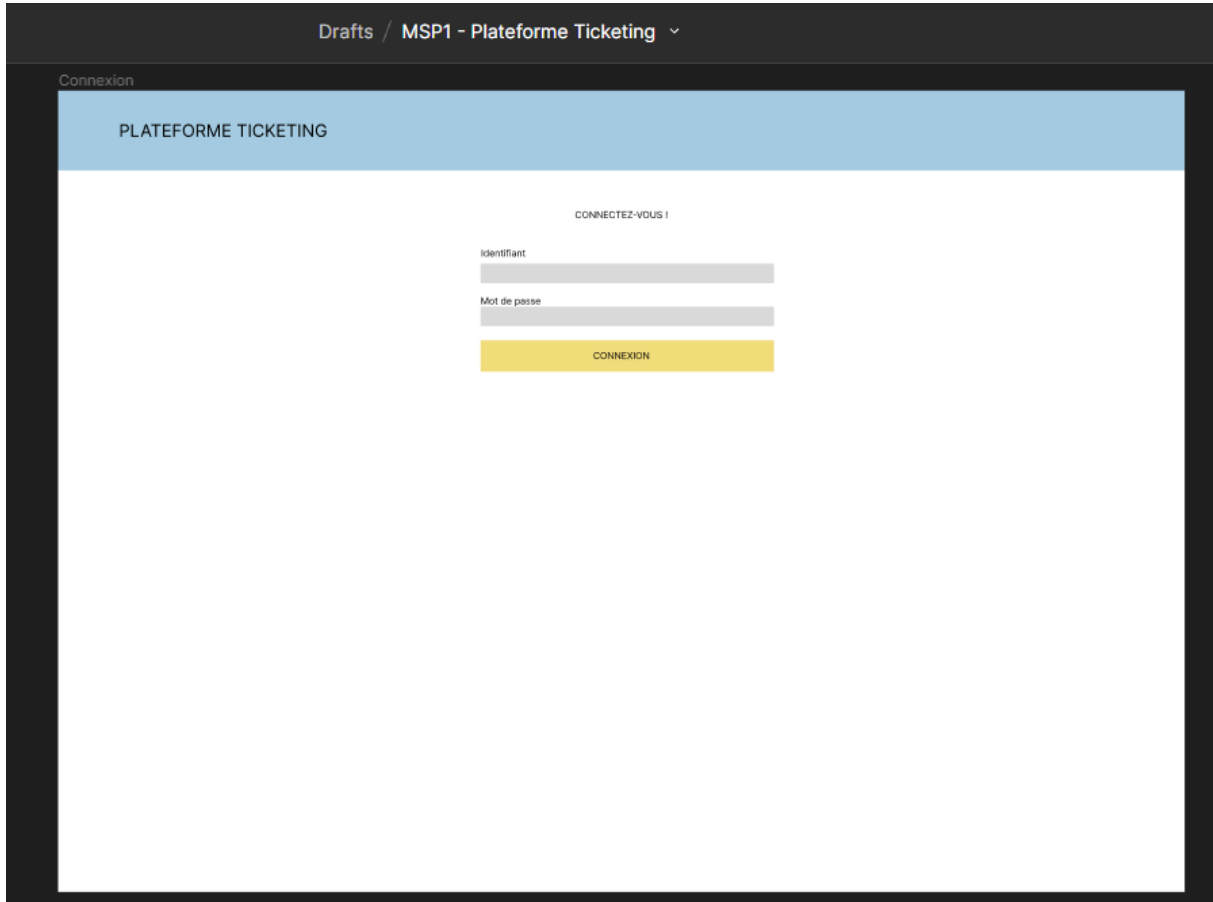
## Diagramme de classe



## Conception et développement de la plateforme

### Connexion à la plateforme

#### Conception



The mockup shows a web interface for a ticketing platform. At the top, a dark header contains the text "Drafts / MSP1 - Plateforme Ticketing" with a dropdown arrow. Below this, a light blue banner reads "Connexion". The main content area has a light blue header "PLATEFORME TICKETING". The login form is centered and includes the text "CONNECTEZ-VOUS !", followed by input fields for "Identifiant" and "Mot de passe", and a yellow "CONNEXION" button.

↪ Maquette conçue sur Figma. L'utilisateur est redirigé vers la page de connexion si l'utilisateur n'est pas connecté.

```
class AppAuthenticator extends AbstractLoginFormAuthenticator
{
    use TargetPathTrait;

    public const LOGIN_ROUTE = 'app_login';

    public function __construct(private UrlGeneratorInterface $urlGenerator)
    {
    }

    /**
     * Fonction dans l'interface qui permet de créer un passport avec l'utilisateur, son mot de passe et un jeton de sécurité
     * @see AuthenticatorInterface
     */
    public function authenticate(Request $request): Passport
    {
        $username = $request->request->get('username', '');

        $request->getSession()->set(Security::LAST_USERNAME, $username);

        return new Passport(
            new UserBadge($username),
            new PasswordCredentials($request->request->get('password', '')),
            [
                new CsrfTokenBadge('authenticate', $request->request->get('_csrf_token')),
                new RememberMeBadge(),
            ]
        );
    }
}
```

↪ Méthodes qui permettent d'exécuter un traitement après une authentification.

```
class MainController extends AbstractController
{
    /**
     * Route principale -> Vérifie si on est connecté et redirige l'utilisateur vers la vue
     * @param Request $request
     * @param ManagerRegistry $managerRegistry
     * @return Response
     */
    #[Route('/', name: 'app_main')]
    public function index(Request $request, ManagerRegistry $managerRegistry): Response
    {
        if (!$this->getUser()) return $this->redirectToRoute("app_login"); //Redirige l'utilisateur vers la page de connexion s'il n'est pas
        $serviceLst = $managerRegistry->getRepository(Service::class)->findAll(); //Sert pour le mode admin
        $status = $request->query->has('status') ? str_replace('_', ' ', $request->query->get('status')) : 'EN ATTENTE'; //Système de filtre
        $dashboard = false;
        if (in_array("ROLE_ADMIN", $this->getUser()->getRoles())) { //Permet d'afficher ou non le dashboard en administrateur
            $dashboard = true;
            if ($request->query->has('dashboard')) {
                $dashboard = $request->query->getBoolean('dashboard');
            }
        }
        return $this->render('index.html.twig', [
            'serviceLst' => $serviceLst,
            'status' => $status,
            'dashboard' => $dashboard
        ]); //On retourne la vue à afficher
    }
}
```

↪ Route par défaut, on redirige l'utilisateur vers la page de connexion s'il n'est pas connecté.

**Connectez-vous !**

Identifiant

Mot de passe

CONNEXION

[Mot de passe oublié ?](#)

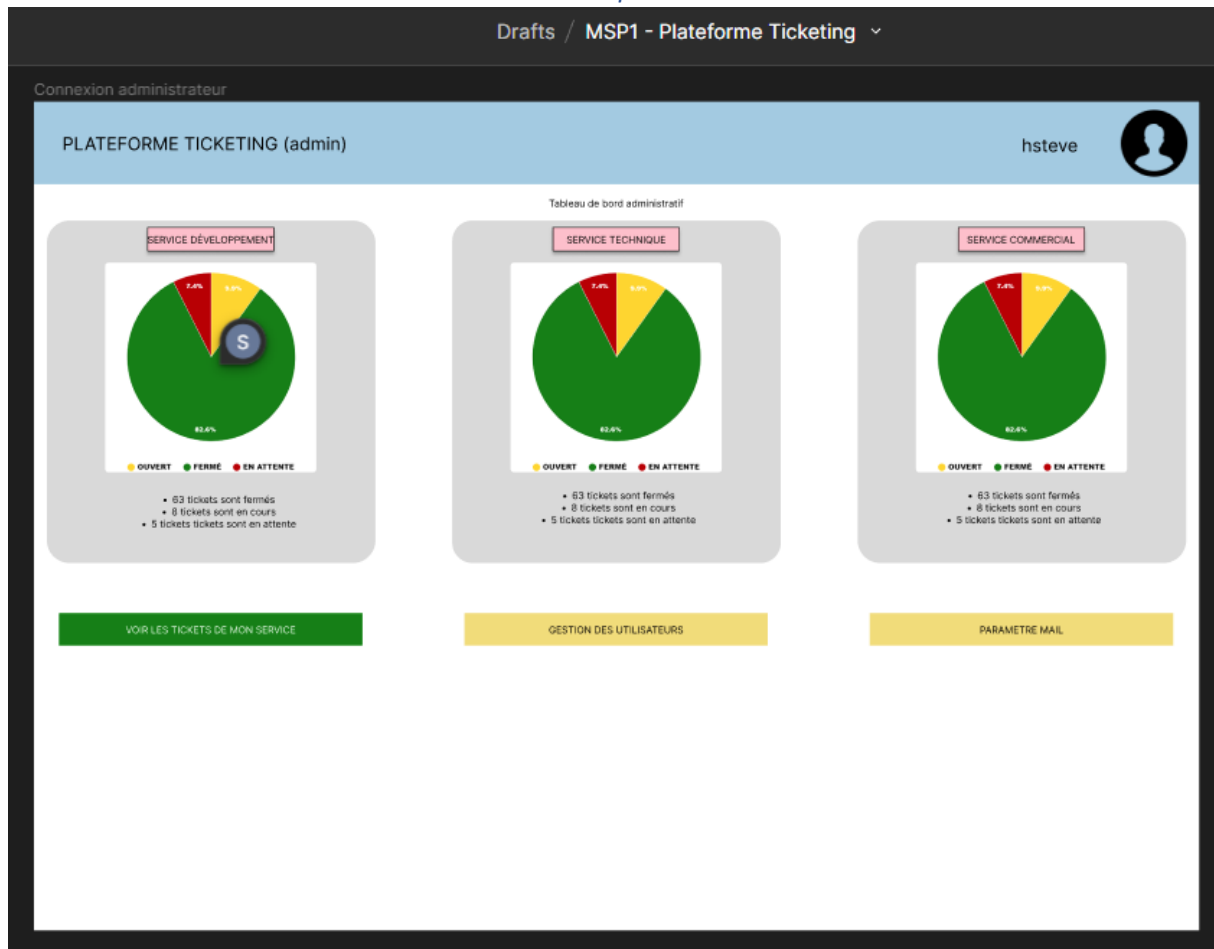
↪ L'identifiant est envoyé par mail avec le mot de passe lors de la création du compte.

Spécificité :

- Un compte administrateur par défaut est créé et envoyé par un email par défaut lors du premier lancement de la page de connexion.
- Un lien « Mot de passe oublié ? » est disponible pour avoir un mot de passe fort générer automatiquement et envoyé par mail si le compte existe.

## Dashboard pour les administrateurs

### Conception



↪ Maquette conçue sur Figma.

```
/**
 * Route principale -> Vérifie si on est connecté et redirige l'utilisateur vers la vue
 * @param Request $request
 * @param ManagerRegistry $managerRegistry
 * @return Response
 */
#[Route('/', name: 'app_main')]
public function index(Request $request, ManagerRegistry $managerRegistry): Response
{
    if (!$this->getUser()) return $this->redirectToRoute("app_login"); //Redirige l'utilisateur vers la page de connexion s'il n'est pas
    $serviceLst = $managerRegistry->getRepository(Service::class)->findAll(); //sert pour le mode admin
    $status = $request->query->has('status') ? str_replace('_', ' ', $request->query->get('status')) : 'EN ATTENTE'; //Système de filtre
    $dashboard = false;
    if (in_array("ROLE_ADMIN", $this->getUser()->getRoles())) { //Permet d'afficher ou non le dashboard en administrateur
        $dashboard = true;
        if ($request->query->has('dashboard')) {
            $dashboard = $request->query->getBoolean('dashboard');
        }
    }
    return $this->render('index.html.twig', [
        'serviceLst' => $serviceLst,
        'status' => $status,
        'dashboard' => $dashboard
    ]); //On retourne la vue à afficher
}
```

↳ Route par défaut, retourne une vue avec des paramètres spécifiques si c'est un administrateur

```
{% if "ROLE_ADMIN" in app.user.roles %}
{#      Mode admin      #}
{% if dashboard == true %}
{#      Affichage du dashboard      #}
<h1 class="text-center font-extrabold m-3">Tableau de bord administratif</h1>
<div class="grid grid-cols-3 gap-4">
    {% for service in serviceLst %}{# on parcourt la liste des services donnés en paramètre lors du rendu de la vue depuis le contrôleur #}
    <div class="bg-gray-300 rounded-lg p-4">
        <a href="{{ path('app_historic_by_service', {'service': service.id}) }}">{# Possibilité de cliquer pour avoir + d'infos #}
        <div class="text-center">
            <span class="badge badge-secondary">{{ service.name }}</span>
        </div>
        <div class="chart-container" style="position: relative; height: 40vh; width: 80vw; margin-left: 20px">
            <canvas id="canvaService{{ service.id }}"></canvas> {# canvas qui se génère après le chargement du contenu #}
        </div>
        <div class="mt-1 grid grid-cols-3 justify-center justify-items-center">
            <span id="waitingBadgeIdService{{ service.id }}" class="badge badge-error"></span>{# Nombre de tickets en attente #}
            <span id="inProgressBadgeIdService{{ service.id }}" class="badge badge-info"></span>{# Nombre de tickets en cours #}
            <span id="closeBadgeIdService{{ service.id }}" class="badge badge-success"></span>{# Nombre de tickets fermés #}
        </div>
    </a>
    </div>
    {% endfor %}
</div>
```

↳ Vérification dans la vue lorsqu'un utilisateur est connecté (index.html.twig) qui montre le tableau de bord si le paramètre dashboard est activé (dashboard == true) et les statistiques par service.

La balise « canvas » qui fera apparaître l'élément graphique d'un camembert généré à partir de ChartJS grâce à son identifiant « canvaService{{ service.id }} » (voir ci-dessous)



```
<script>
    document.addEventListener("DOMContentLoaded", () => {
        let ctx;
        let chart;
        let ms = [];
        {% for service in serviceList %}/*On parcourt la liste des services*/
        ctx = document.getElementById('canvaService{{ service.id }}');

        //Création de la requête AJAX en POST
        $.ajax({
            type: 'POST',
            url: '{{ path('app_api_count_tickets_service', {'service':service.id}) }}',
            data: { '_csrf_token': "{{ csrf_token('api-count~service.id') }}" },
            success: function (response) { //s'il n'y pas d'erreur
                if (response !== 500) { //la route peut retourner new JsonResponse(500)
                    ms["{{ service.id }}"].data.datasets[0].data[0] = parseFloat(response[0]); //Set du data par rapport à la réponse d
                    ms["{{ service.id }}"].data.datasets[0].data[1] = parseFloat(response[1]);
                    ms["{{ service.id }}"].data.datasets[0].data[2] = parseFloat(response[2]);
                    //tableau de longueur 3 car il y a 3 types de tickets
                    ms["{{ service.id }}"].update();
                    console.log(response);
                    $('#waitingBadgeIdService{{ service.id }}').html("Ticket EN ATTENTE: " + response[0]); //Mise à jour du badge qui af
                    $('#inProgressBadgeIdService{{ service.id }}').html("Ticket EN COURS: " + response[1]);
                    $('#closeBadgeIdService{{ service.id }}').html("Ticket FERMÉ: " + response[2]);
                } else {
                    console.error('Erreur lors de la requête. (500)');
                }
            },
            error: function () { //il peut avoir une erreur
                console.error('Erreur lors de la requête. ');
            }
        });

        //Création d'un diagramme
        chart = new Chart(ctx, {
            type: 'doughnut',
            data: {
                labels: [],
                datasets: [{
                    label: '',
                    data: [1, 1, 1], //défaut 3 types de valeur 1, c'est le poids que ça prendra dans le diagramme
                    backgroundColor: [
                        'rgb(255, 99, 132)',
                        'rgb(54, 162, 235)',
                        'rgb(123,255,86)'
                    ],
                    hoverOffset: 2
                }],
            },
        });
        ms["{{ service.id }}"] = chart;
        {% endfor %}
        console.log(ms);
    });
</script>
```

↳ Pour mettre à jour les balises « canvas », on utilise l'évènement du chargement du DOM et on met à jour, pour chaque service, le camembert de ChartJS associé en faisant des appels AJAX à l'aide de JQuery à la route « app\_api\_count\_tickets\_service »

Spécificité :

- La requête AJAX à l'aide de JQuery est sécurisée car elle utilise la méthode POST et un jeton « \_csrf\_token »

```
/**
 * @param Request $request
 * @param EntityManagerInterface $manager
 * @param $service
 * @return JsonResponse
 */
#[Route('/api/count-tickets/{service}', name: 'app_api_count_tickets_service', methods: ['POST'])]//on tolère uniquement les requêtes
public function apiCount(Request $request, EntityManagerInterface $manager, $service): JsonResponse
{
    if ($request->request->has('_csrf_token') && $this->isCsrfTokenValid('api-count' . $service, $request->request->get('_csrf_token'))
        $service = $manager->getRepository(Service::class)->find($service);
    if ($service != null) {
        $nClose = 0;
        $nInProgress = 0;
        $nWaiting = 0;
        $tickets = $manager->getRepository(Ticket::class)->findBy(['service'=>$service]);
        foreach ($tickets as $ticket){
            //on récupère les tickets qui sont fermés (ayant une date de résultat) ou dont il y a traitements et que le status du
            dd($ticket->getTreatments(),sizeof($ticket->getTreatments()));
            if(sizeof($ticket->getTreatments()) == 0) {
                $nWaiting += 1;
            }else if($ticket->getTreatments()->last()->getStatus() == "EN ATTENTE" || str_contains($ticket->getTreatments()->last(
                //les tickets qui n'ont pas encore de traitement sont par défaut en attente, on traite le comptage de ces tickets
                $nWaiting += 1;
            )else if($ticket->getTreatments()->last()->getStatus() == "EN COURS"){
                $nInProgress += 1;
            }else if($ticket->getResultDate()!=null || $ticket->getTreatments()->last()->getStatus() == "Fermé"){
                $nClose+=1;
            }
        }

        dump(array($nWaiting, $nInProgress, $nClose));
        return new JsonResponse(array($nWaiting, $nInProgress, $nClose));//on retourne un JsonResponse d'un array de chaque compte
    }
    return new JsonResponse(500);
}
```

↳ la route « app\_api\_count\_tickets\_service » qui tolère uniquement les requêtes POST et qui sert à avoir sous forme de JsonResponse, chaque compteur pour les traitements de tickets.

```
#[IsGranted('ROLE_ADMIN')]
#[Route('/admin/historic')]
class HistoricController extends AbstractController
```

```
#[IsGranted("ROLE_ADMIN")]
#[Route('/admin/gestion/utilisateur')]
class UserController extends AbstractController
{
```

↳ Certaines routes sont protégées ↔

## Dashboard final



↪ En cliquant sur un service l'administrateur aura un historique des tickets par rapport au service (voir ci-dessous).

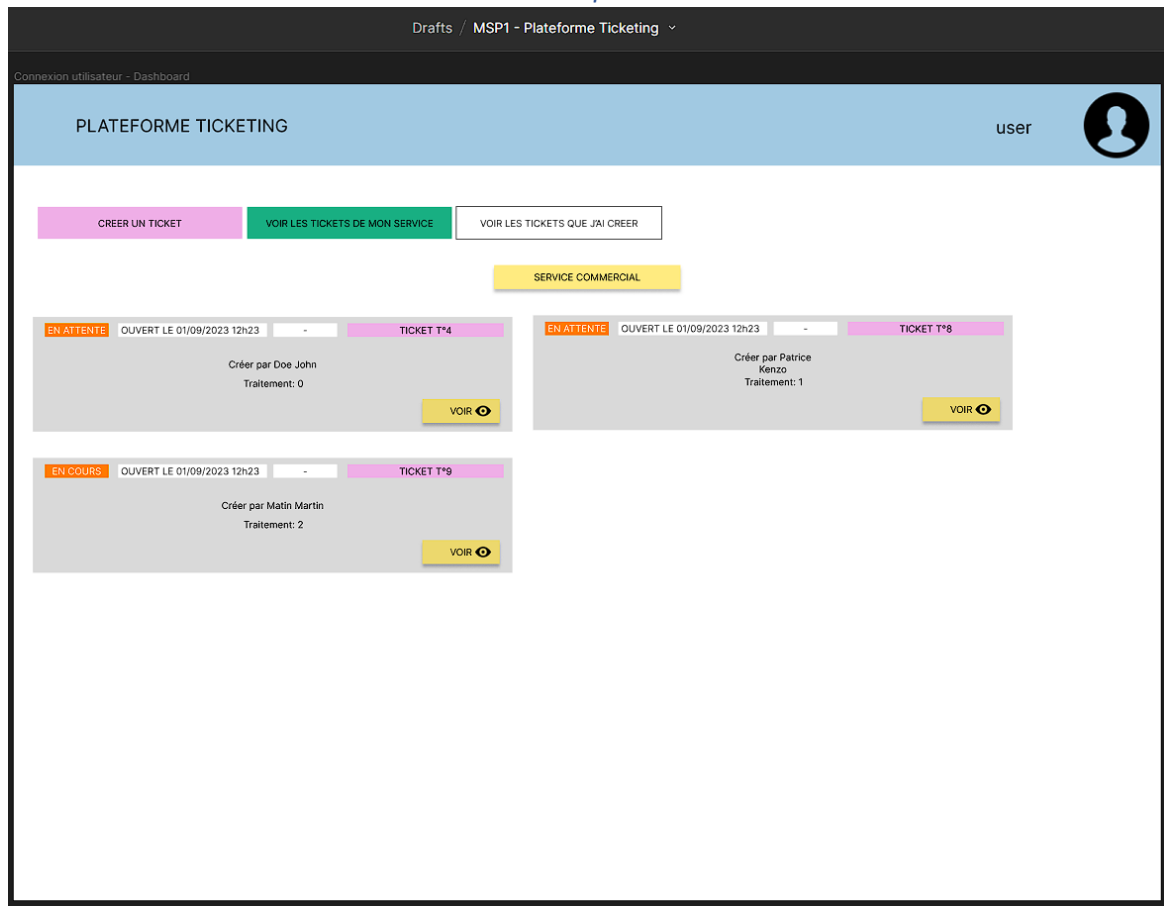
Plateforme Ticketing (admin) hsteve

**Historique**  
SERVICE TECHNIQUE

<p><b>Fermé</b> Ouvert le: 31/08/2023 10:56 - Fermé le: 31/08/2023 16:14 - <b>TICKET T*2</b></p> <p>Créer par Doe John Traitement: 1</p> <p><a href="#">VOIR</a></p>	<p><b>Fermé</b> Ouvert le: 31/08/2023 10:58 - Fermé le: 31/08/2023 16:12 - <b>TICKET T*3</b></p> <p>Créer par Doe John Traitement: 4</p> <p><a href="#">VOIR</a></p>	<p><b>TRANSFÈRE // EN ATTENTE</b> Ouvert le: 31/08/2023 13:17 - <b>TICKET T*4</b></p> <p>Créer par Doe John Traitement: 2</p> <p><a href="#">VOIR</a></p>
--	--	---

[RETOUR](#)

## Page utilisateur *Conception*

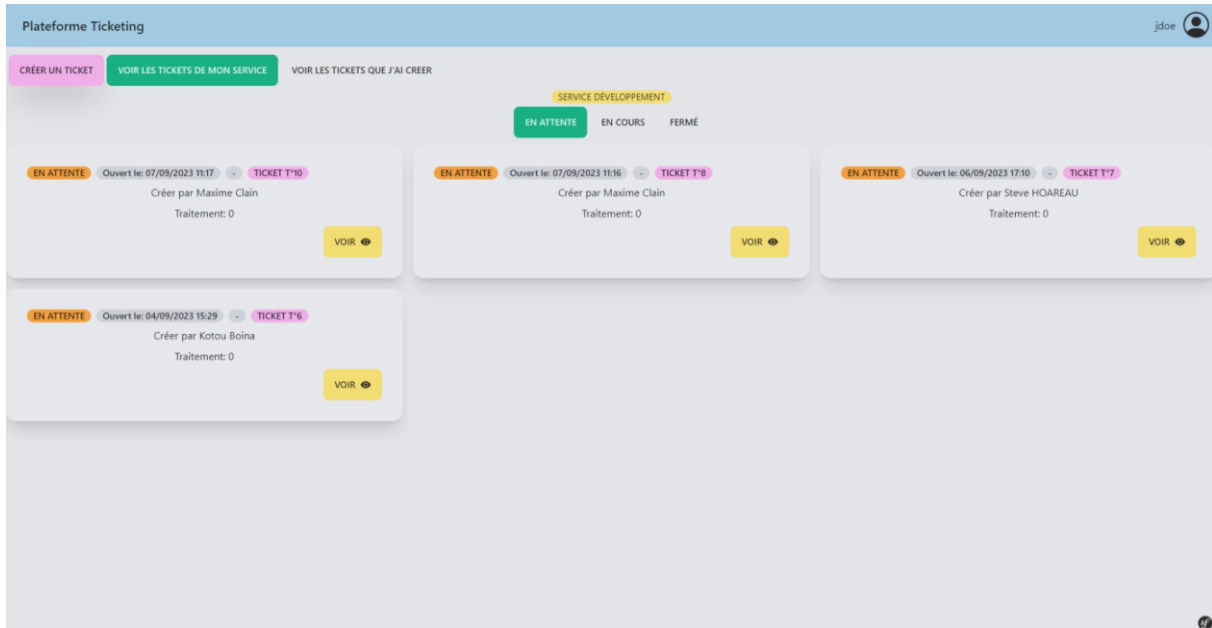


↪ Maquette conçue sur Figma.

```
[% else %]<!-- l'utilisateur n'est pas ADMIN --> Mode user </-->
<div>
<a class="btn btn-secondary shadow-xl ml-1" href="{( path('app_ticket_now') )}">CRÉER UN TICKET</a>
<a class="btn btn-success" href="{( path('app_main') )}">VOIR LES TICKETS DE MON SERVICE</a>
<a class="btn btn-ghost" href="{( path('app_main_my_tickets') )}">VOIR LES TICKETS QUE J'AI CRÉÉS</a>
<div class="text-center">
<div class="badge badge-primary">SERVICE {( app.user.service.name )}</div>
<div>
{%- if status == "EN ATTENTE" -%}
    <a class="btn btn-success" href="{( path('app_main') )}">EN ATTENTE</a>
{%- else %}
<a class="btn btn-ghost" href="{( path('app_main') )}">EN ATTENTE</a>
{%- endif %}
{%- if status == "EN COURS" -%}
    <a class="btn btn-success" href="{( path('app_main') )}">status=EN_COURS</a>
{%- else %}
    <a class="btn btn-ghost" href="{( path('app_main') )}">status=EN_COURS</a>
{%- endif %}
{%- if status == "Formé" -%}
    <a class="btn btn-success" href="{( path('app_main') )}">status=formé</a>
{%- else %}
    <a class="btn btn-ghost" href="{( path('app_main') )}">status=formé</a>
{%- endif %}
</div>
</div>
<div class="grid grid-cols-1 mt-3 gap-4">
{%- set ticket_in_cat = false %}{% variable qui va servir à afficher ou pas le message de filtrage (instruction en bas) %}
{%- set status in app.user.service.ticket %}
{%- # status est la catégorie qui filtre par status et qui est définie en requête GET -# %}
{%- if status == "EN ATTENTE" or (ticket.treatments.last != false and status in ticket.treatments.last.status) %}
{%- set ticket_in_cat = true %}
<div class="card bg-gray-200 shadow-xl">
<div class="card-body">
<h2 class="card-title">
<span class="badge badge-warning">
{%- if ticket.treatments.last == false or ticket.treatments.last.status is null %}
EN ATTENTE
{%- else %}
{%- if (ticket.treatments.last.status) %% le ticket récupère le statut du dernier traitement %}
<span class="badge badge-ghost"><div>le: {( ticket.createdAt|date('d/M/Y H:i') )}</div>
<span class="badge badge-ghost">{( if ticket.resultDate is not null %}Formé le: {( ticket.resultDate|date('d/M/Y H:i') )}</span>
<span class="badge badge-secondary">TICKET {( ticket.id )}</span>
<p>Créer par {( ticket.creator.firstName ~ " " ~ ticket.creator.name )}</p>
<p>Traitement: {( ticket.treatments|length )}</p>
<div class="card-actions justify-end">
<a class="btn btn-primary" href="{( path('app_ticket_see') / 'id': ticket.id )}">voir
<svg xmlns="http://www.w3.org/2000/svg" width="25px" height="16px" fill="currentcolor" class="bi bi-eye">
<path d="M0 0 25 0 25 25 0 25 0 0 z" />
<path d="M0 0 25 0 25 25 0 25 0 0 z" />
</svg>
</a>
</div>
</div>
{%- endif %}
{%- endfor %}
{%- if not ticket_in_cat %}( ... )
Il n'y a pas de ticket dans cette catégorie...
</div>
</div>
</div>
</div>
```

↪ Vue appelée depuis la route par défaut qui a été montrer plus haut. Chaque ticket récupère par défaut le statut 'EN ATTENTE' ou le statut de son dernier traitement dans le cas échéant.

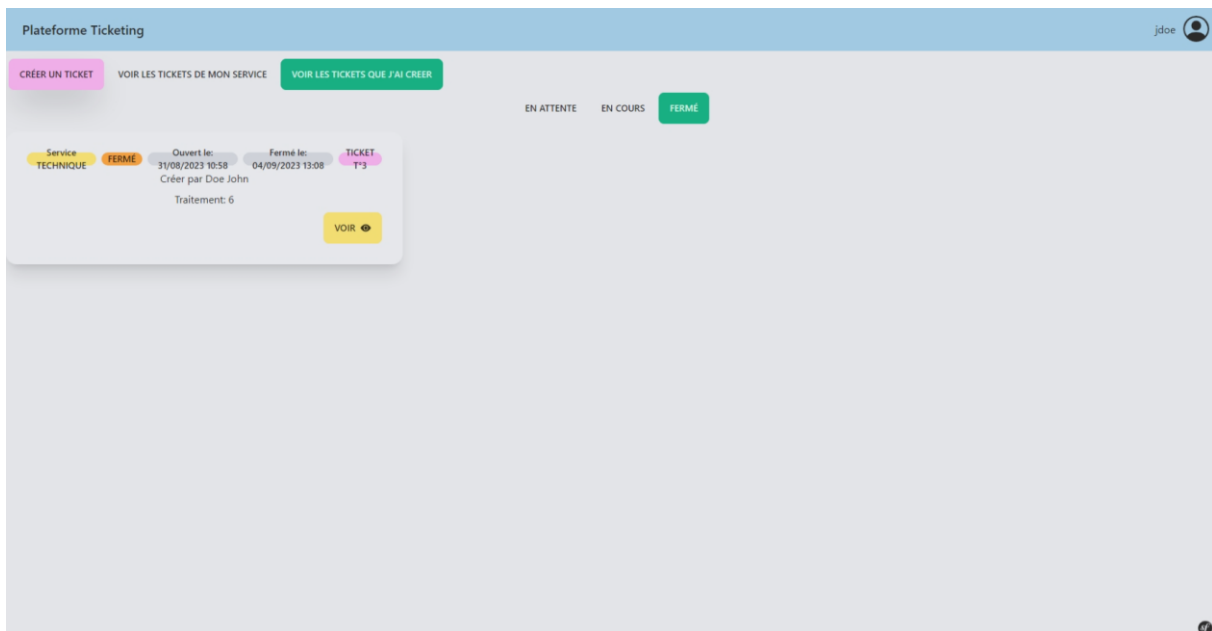
## Page utilisateur finale



The screenshot shows the 'Plateforme Ticketing' interface. At the top, there are navigation tabs: 'CRÉER UN TICKET', 'VOIR LES TICKETS DE MON SERVICE' (selected), and 'VOIR LES TICKETS QUE J'AI CRÉER'. Below these, there's a filter for 'SERVICE DÉVELOPPEMENT' with sub-filters 'EN ATTENTE' (selected), 'EN COURS', and 'FERMÉ'. The main area displays a list of tickets in 'EN ATTENTE' status:

- TICKET T\*10**: Ouvert le: 07/09/2023 11:17, Créé par Maxime Clain, Traitement: 0. [VOIR]
- TICKET T\*9**: Ouvert le: 07/09/2023 11:16, Créé par Maxime Clain, Traitement: 0. [VOIR]
- TICKET T\*7**: Ouvert le: 06/09/2023 17:10, Créé par Steve HOAREAU, Traitement: 0. [VOIR]
- TICKET T\*6**: Ouvert le: 04/09/2023 15:29, Créé par Kotou Boina, Traitement: 0. [VOIR]

↳ L'utilisateur a accès à ses tickets avec un système de filtre par statut. Il peut aussi voir les tickets qu'il a créé(voir-ci-dessous).



The screenshot shows the 'Plateforme Ticketing' interface with the 'VOIR LES TICKETS QUE J'AI CRÉER' tab selected. The filter for 'SERVICE DÉVELOPPEMENT' is still present, but the sub-filters are 'EN ATTENTE', 'EN COURS', and 'FERMÉ' (selected). The main area displays a single ticket in 'FERMÉ' status:

- TICKET T\*3**: Service: TECHNIQUE, Ouvert le: 31/08/2023 10:58, Fermé le: 04/09/2023 13:08, Créé par Doe John, Traitement: 6. [VOIR]

[RETOUR](#)

Ouvert le 31/08/2023 10:58

Fermer le 04/09/2023 13:08

TICKET T3

Créer par Doe John

Traitement: 6

Raison d'ouverture: Le client à besoin d'une mise à jour office

Fermé par: Kotou Boina

Raison de fermeture: mis à jour

RELAYE

Commencé le 31/08/2023 14:54

Fin le 31/08/2023 14:55

Traité par Doe John

Observations: Le client n'est pas en contrat

RELAYE

Commencé le 31/08/2023 14:58

Fin le 31/08/2023 15:02

Traité par Maxime Clain

Observations: Le client n'a pas le logiciel

RELAYE

Commencé le 31/08/2023 15:04

Fin le 31/08/2023 15:57

Traité par Doe John

Observations: C'est fait

FERME

Commencé le 31/08/2023 15:57

Fin le 31/08/2023 16:12

Traité par Kotou Boina

Observations: pris en charge

RELAYE

Commencé le 04/09/2023 13:05

Fin le 04/09/2023 13:08

Traité par Steve HOAREAU

Observations: ce n'est pas la bonne version...

FERME

Commencé le 04/09/2023 13:08

Fin le 04/09/2023 13:08

Traité par Kotou Boina

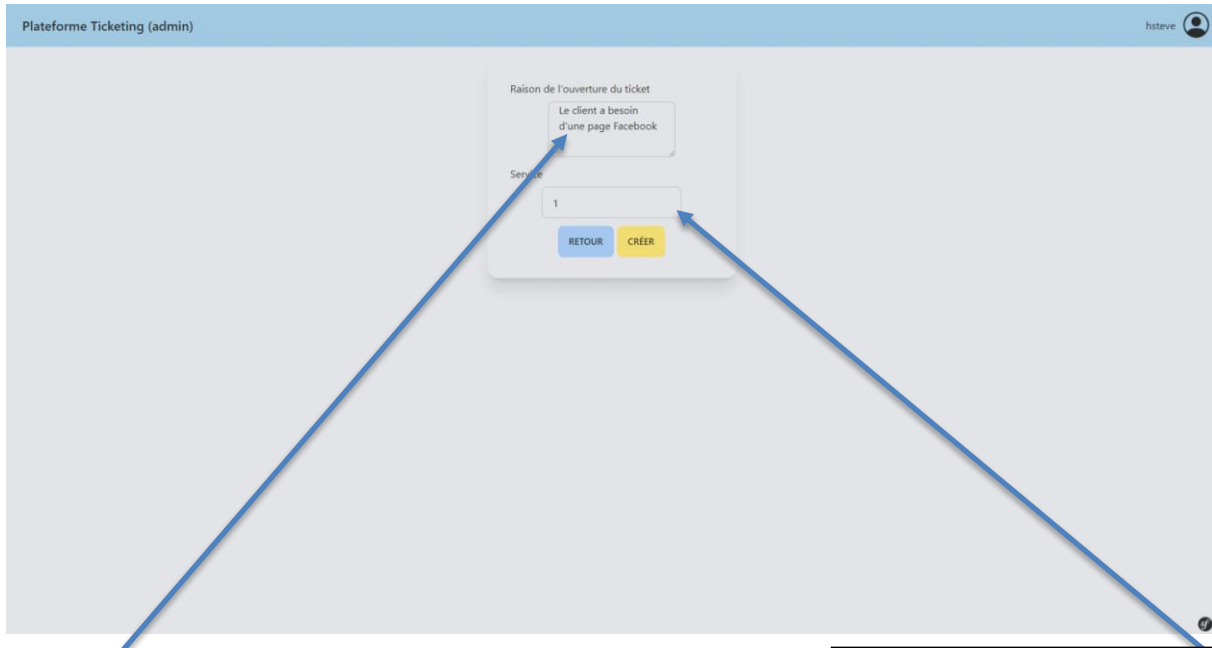
Observations: Mis à jour

↪ L'utilisateur a accès aux traitements du ticket.

## VI. Présentation du jeu d'essai élaboré par le candidat de la fonctionnalité la plus représentative (données en entrée, données attendues, données obtenues)

Création de ticket

Données en entrées



Une raison d'ouverture (obligatoire)

L'identifiant d'un service (obligatoire)

Données attendues

- Une raison d'ouverture non vide
- Un identifiant d'un service non vide et existant
- L'utilisateur connecté qui est remonté automatiquement dans le code



## Données obtenues

id	service_id	creator_id	create_date	problem	result	result_date	transferred
12	1	1	2023-09-07 17:09:58	Le client a besoin d'une page Facebook	NULL	NULL	0

↪ Le ticket a été créé et ajouté dans la base de données

Le champ « id » est l'identifiant du ticket.

Le champ « service\_id » est l'identifiant du service non vide et existant que l'utilisateur a renseigné

Le champ « create\_date » est la date quand l'utilisateur a créé qui est définie automatique dans le code par la date du jour.

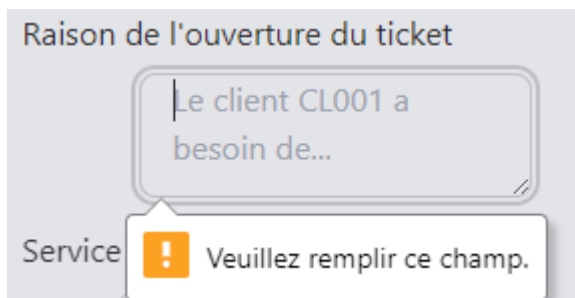
Le champ « problem » est la raison d'ouverture non vide que l'utilisateur a définie quand il a créé le ticket.

Le champ « result » et « result\_date » sont NULL par défaut et sert pour la fermeture du ticket.

Le champ « transferred » est false par défaut (0) et sert à définir quand le ticket est transféré.

En cas d'erreur :

- Pour une raison d'ouverture vide :



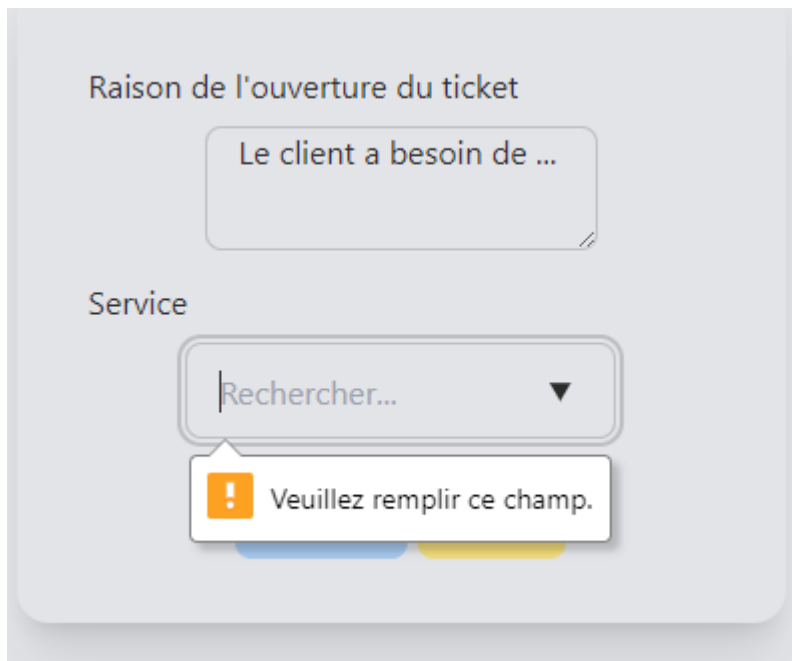
Raison de l'ouverture du ticket

Le client CL001 a besoin de...

Service

! Veuillez remplir ce champ.

- Pour un service vide :



Raison de l'ouverture du ticket

Le client a besoin de ...

Service

Rechercher...

! Veuillez remplir ce champ.

- Pour un service non existant :

## Modification d'un utilisateur (admin seulement)

### Données en entrées

### Données attendues

- Un nom d'utilisateur non vide et non existant (sauf si le nom d'utilisateur est inchangé)
- Une adresse e-mail non vide et de type e-mail et non existant (sauf si l'adresse e-mail est inchangée)
- Un nom non vide
- Un prénom non vide
- L'identifiant d'un service non vide et existant

## Données obtenues

id	service_id	email	username	roles (DC2Type=json)	password	name	firstname	active
2	1	jdoe@gmail.com	jdoe	["ROLE_USER"]	\$2y\$13\$mDlrZZjKFNlo3Wv.8qvhs.0A0dN4INnwRmmH9aGQC8H...	John	Doe	1

## ↪ Avant la modification

id	service_id	email	username	roles (DC2Type=json)	password	name	firstname	active
2	1	john.doe@gmail.com	jdoe	["ROLE_USER"]	\$2y\$13\$mDlrZZjKFNlo3Wv.8qvhs.0A0dN4INnwRmmH9aGQC8H...	John	Doe	1

## ↪ Après la modification

En cas d'erreur :

- Pour un nom d'utilisateur vide

Nom d'utilisateur

jdoe

E-Mail

! Veuillez remplir ce champ.

- Pour un nom d'utilisateur déjà existant

Plateforme Ticketing (admin) hsteve

⚠ Ce nom d'utilisateur est déjà pris...

**Modifier un utilisateur**

Nom d'utilisateur  
hsteve

E-Mail  
john.doe@gmail.com

Nom  
John

Prénom  
Doe

Service  
1

RETOUR METTRE À JOUR

- Pour une adresse mail vide

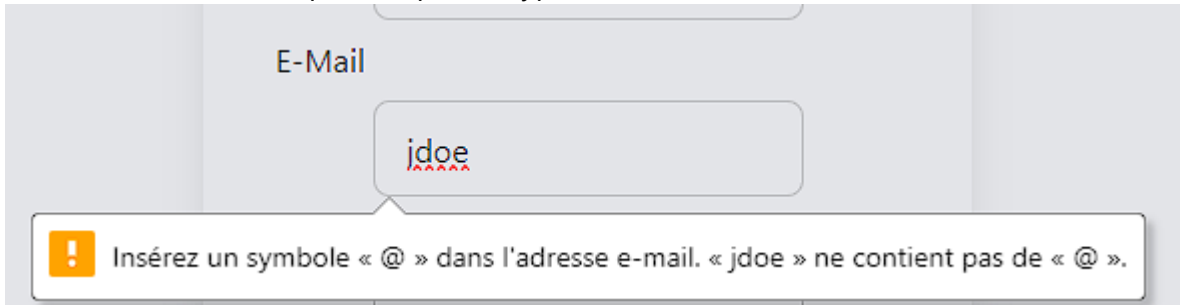
E-Mail

jdoe@gmail.com

Nom

! Veuillez remplir ce champ.

- Pour une adresse mail qui n'est pas du type e-mail

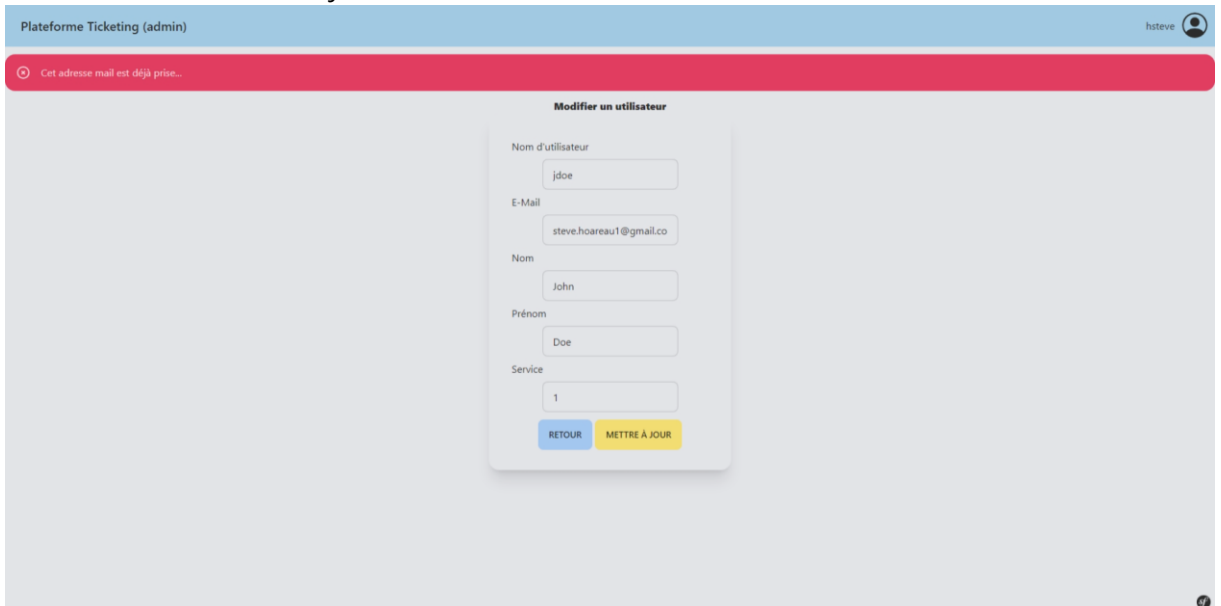


E-Mail

jdoue

❗ Insérez un symbole « @ » dans l'adresse e-mail. « jdoue » ne contient pas de « @ ».

- Pour une adresse mail déjà existante



Plateforme Ticketing (admin) hsteve

ⓘ Cet adresse mail est déjà prise...

**Modifier un utilisateur**

Nom d'utilisateur  
jdoue

E-Mail  
steve.hoareau1@gmail.co

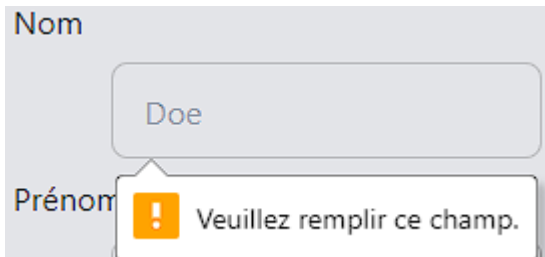
Nom  
John

Prénom  
Doe

Service  
1

[RETOUR](#) [METTRE À JOUR](#)

- Pour un nom vide



Nom

Doe

Prénom

❗ Veuillez remplir ce champ.

- Pour un prénom vide



Prénom

John


Service

❗ Veuillez remplir ce champ.


- Pour un identifiant de service vide


Service

Rechercher...

 Veuillez remplir ce champ.

- Pour un service non existant

Plateforme Ticketing (admin) hsteve 

 Ce service n'existe pas...

**Modifier un utilisateur**

Nom d'utilisateur

E-Mail

Nom

Prénom

Service

[RETOUR](#) [METTRE À JOUR](#)

## VII. Description de la veille, effectuée par le candidat durant le projet, sur les vulnérabilités de sécurité

Toutes les sources de la veille de sécurité sur les vulnérabilités de la plateforme ont été vérifiées sur le site <https://owasp.org/Top10>.

### A01:2021 – Contrôles d'accès défaillants

Pour réduire la vulnérabilité d'accès défaillants (par changement d'url, par api, sans authentification), on a rajouté une condition dans chacune des routes de Symfony (sauf login) pour vérifier si l'utilisateur qui fait la requête est identifié.

Pour le changement d'url, Symfony traite par défaut une URL invalide. Toutes les requêtes passées en URL depuis la plateforme sont des requêtes qui touchent uniquement au côté client (modification visuelle) mais aucune ne fait la persistance des données.

Les routes les plus sensibles de Symfony sont sécurisées par une vérification des droits de l'utilisateur par rapport à son rôle, dans ces routes, certaines utilisent aussi les jetons d'accès (CSRF Token) pour garantir la sécurité (comme les routes pour ajouter, modifier ou supprimer des éléments de la plateforme tels que des utilisateurs ou la mise à jour des tickets et de leur traitements).

### A02:2021 – Défaillances cryptographiques

Pour la vulnérabilité des défaillances cryptographiques il n'y a eu aucun traitement car les serveurs utilisés sont consacrés uniquement pour le développement et non la mise en production du projet.

### A03:2021 – Injection

L'injection SQL est traitée par défaut par Symfony (composant Doctrine) en utilisant des requêtes préparées pour chaque traitement sur la base de données.

Aucune donnée peut être persistée depuis JavaScript.

### A04:2021 – Conception non sécurisée

Le code de la plateforme est un code compilé, testé et fonctionnel.

### A05:2021 – Mauvaise configuration de sécurité

La base de données peut être vulnérable si l'utilisateur a accès au fichier permettant de configurer la liaison entre la plateforme et la base de données. Cette faille peut être traitée en utilisant un utilisateur de base de données ayant le moins de droit que possible et en délocalisant la connexion dans un autre dossier ou en changeant les droits d'accès aux fichiers.

### A06:2021 – Composants vulnérables et obsolètes

Tous les composants sont à jour et ont été vérifiés par leurs auteurs (avec test d'intégrité, test unitaire avec une plateforme telle que Jenkins), mais on peut toujours trouver une faille.

En conclusion, la plateforme respecte la plupart des règles de sécurité et empêche-la plupart des failles. Les failles restantes doivent être réglées avant la mise en production finale du projet.

Le projet contient des fichiers de journalisation permettant d'avoir un historique des requêtes.

## VIII. Description d'une situation de travail ayant nécessité une recherche, effectuée par le candidat durant le projet

Dans le cadre de travail de mon entreprise, on a eu besoin de faire une plateforme pour les tickets en interne pour avoir les mêmes fonctionnalités que la présente plateforme. Je n'ai pas pu faire ce projet en entreprise vu que j'étais déjà sur d'autre projet, je l'ai donc pris pour le concevoir, le développer et le présenter à travers ce document et le projet qui en découle.

Les recherches effectuées sont des recherches sur internet pour le débogage de plateforme.