

# PLATEFORME TICKETING

23 août

2024

Dossier projet – La conception, le développement et le  
déploiement d'une application sécurisée organisée en couches



## Table des matières

<b>I. Liste des compétences du référentiel qui sont couvertes par le projet</b>	<b>3</b>
<b>II. Cahier des charges, expression des besoins, ou spécifications fonctionnelles du projet</b>	<b>4</b>
<b>III. Environnement technique</b>	<b>6</b>
<b>IV. Gestion de projet</b>	<b>6</b>
Installation et configuration de l'environnement de travail:	8
Cas d'utilisation:	11
<b>V. Réalisations permettant la mise en œuvre des compétences</b>	<b>12</b>
Conception des maquettes:	12
Enchaînement de maquette des interfaces:	13
Modèle conceptuel des données:	14
Modèle physique des données:	15
Diagramme de classe:	17
Conception et développement de la connexion utilisateur:	19
Conception et développement des fonctionnalités pour les utilisateurs:	19
Jeu d'essai:	21
Tests unitaires:	23
Visuel final:	24
Conception et développement des fonctionnalités pour les administrateurs:	24
Jeu d'essai:	27
Tests d'intégrations:	28
Visuel Final:	29
Affichage des statistiques:	30
Jeu d'essai:	32
Plan de tests:	32
Mise en production dans une démarche DevOps	33
Description de la veille, effectuée par le candidat durant le projet, sur les vulnérabilités de sécurité, description des vulnérabilités éventuellement trouvées et des failles potentiellement corrigées:	36
<b>VI. Annexes</b>	<b>38</b>
Annexe 1: Documentation technique de l'environnement de développement:	38
Annexe 2: Interface utilisateur, page de connexion:	40
Annexe 3: Interface utilisateur, création de ticket:	41
Annexe 4: Intégralité du plan de tests:	42
Annexe 5: Fonctions "up()", "createUserAndLogin()" et "createService()" de la classe "TicketControllerTest":	44
Annexe 6: Déclencheur "treatment_after_insert" :	46
Annexe 7: Interface administrateur, tableau de bord	47
Annexe 8: Requête AJAX se trouvant dans la vue "index.html.twig":	48
Annexe 9: Procédure "count_tickets_service":	49
Annexe 10: Workflow GitHub actions:	50

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

Le projet couvre obligatoirement les compétences suivantes :

Développer une application sécurisée	Installer et configurer son environnement de travail en fonction du projet
	Développer des interfaces utilisateur
	Développer des composants métier
	Contribuer à la gestion d'un projet informatique
Concevoir et développer une application sécurisée organisée en couches	Analyser les besoins et maquetter une application
	Définir l'architecture logicielle d'une application
	Concevoir et mettre en place une base de données relationnelle
	Développer des composants d'accès aux données SQL et NoSQL
Préparer le déploiement d'une application sécurisée	Préparer et exécuter les plans de tests d'une application
	Préparer et documenter le déploiement d'une application
	Contribuer à la mise en production dans une démarche DevOps

## II. 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 fictive.

Ce projet est réalisé pendant la formation.

- 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 une raison de création et le service concerné.

- Suivi de ticket

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

- Prise en charge de ticket

- Un ticket peut avoir seulement un seul traitement d'ouvert
- 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 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 utilisateur (nombre de tickets 'Créé', nombre de ticket 'Ouvert', nombre de ticket 'Fermé').

### 4. Fonctionnalité de la base de données

- Une procédure est nécessaire pour récupérer le nombre de tickets par service.
- Une procédure est nécessaire pour récupérer le nombre de tickets par utilisateur.
- Un déclencheur est nécessaire pour mettre à jour un ticket après la mise à jour de son dernier traitement.

- Technologies utilisées

- o Conception de la plateforme

- Maquettes de la plateforme avec Figma.
    - Diagramme de cas d'utilisation avec PlantUml.
    - Modèle conceptuel de données avec Looping.
    - Diagramme de classe avec Dia.

- o Développement de la plateforme

- Back-office en PHP 8.2
    - Front-office en CSS 3

- o Persistance des données

- La base de données avec MySQL 8.2

- Livrable et délais

- o La plateforme complète et fonctionnelle doit être livrée conformément aux spécifications.
  - o La date de livraison prévue pour la plateforme est le 04/09/2024.

### III. Environnement technique

#### 1. Développement de la plateforme

- Le back-office a été codé en PHP avec le Framework Symfony 6.3.7
- Le front-office 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 passe 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 passe des utilisateurs sont hashés 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.
- L'accès à la plateforme se fait en HTTPS

### IV. Gestion de projet

Suivi de projet avec GitHub Project en Tableau.

Lors d'une résolution de bugs, les collaborateurs créés des Issues, le service de développement crée des branches par rapport aux issues.

Lors d'une création de nouvelle fonctionnalité, le lead-développeur crée une Milestone.

Le lead-développeur valide le code et met à jour sur la branche principale.

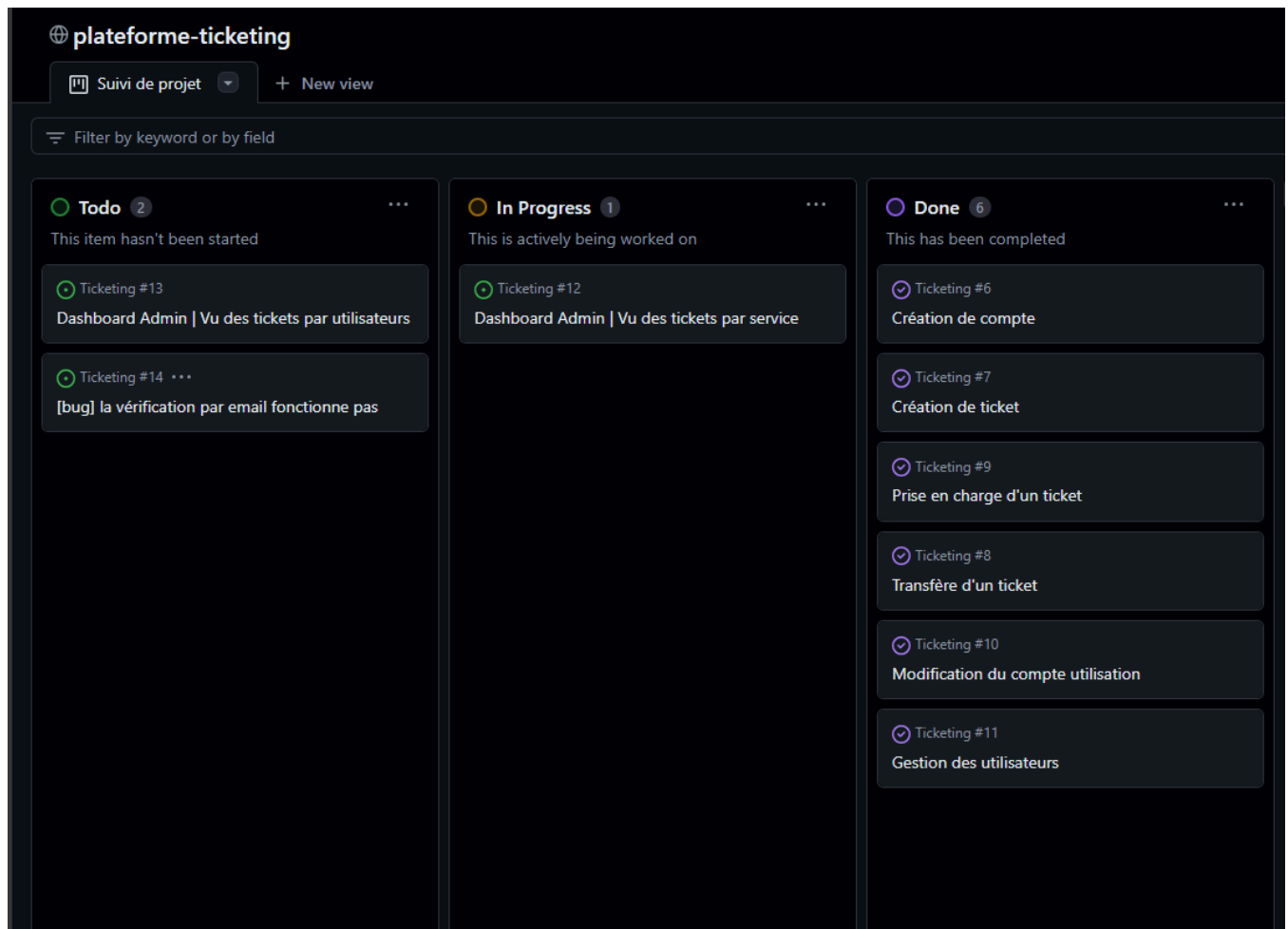


Figure 1: Suivi de projet GitHub project

## Installation et configuration de l'environnement de travail:

L'équipe de développement a accès à un GitHub permettant de récupérer le code.

Une documentation technique est disponible à la racine du projet permettant à un nouveau contributeur à être préparé à développer rapidement. (c.f. Annexe 1)

L'environnement de développement est créé à l'aide d'un docker-compose, et d'un Dockerfile.

```
services:
  mysql:
    image: mysql:8.2
    container_name: ticketing_db
    environment:
      MYSQL_ROOT_PASSWORD: rootpassword
      MYSQL_DATABASE: ticketing
      MYSQL_USER: ticketing
      MYSQL_PASSWORD: ticketing
    command:
      - --log-bin-trust-function-creators=1
    volumes:
      - mysql_data:/var/lib/mysql
    ports:
      - "3306:3306"
  symfony:
    container_name: ticketing_app
    build:
      context: .
      dockerfile: Dockerfile
    volumes:
      - symfony_data:/var/www/symfony
    ports:
      - "8080:8080"
      - "22:22"
    depends_on:
      - mysql
volumes:
  mysql_data:
  symfony_data:
```

Figure 2: Docker-Compose de l'environnement de développement



```
# Utiliser l'image PHP 8.2 CLI officielle
FROM php:8.2-cli

# Installer les dépendances système et les extensions PHP nécessaires
RUN apt-get update && apt-get install -y \
    git \
    unzip \
    libicu-dev \
    libzip-dev \
    nano \
    openssh-server \
    && docker-php-ext-install \
    intl \
    zip \
    pdo_mysql

# Définir le répertoire de travail
WORKDIR /var/www/symfony

# Copier les fichiers de l'application
COPY . .

# Définir le répertoire de travail
WORKDIR /var/www/symfony/web

# Installer Composer
RUN curl -sS https://getcomposer.org/installer | php --
--install-dir=/usr/local/bin --filename=composer

# Installer Node.js 20
RUN curl -fsSL https://deb.nodesource.com/setup_20.x | bash - \
    && apt-get install -y nodejs

# Vérifier les versions de Node.js et npm
RUN node --version && npm --version

# Installer Symfony CLI
RUN curl -sS https://get.symfony.com/cli/installer | bash
RUN mv /root/.symfony5/bin/symfony /usr/local/bin/symfony

# Créer un utilisateur non-root
RUN useradd -ms /bin/bash symfony && \
    echo "symfony:symfony" | chpasswd

RUN chown symfony:symfony /var/www/symfony -R

# Configurer SSH pour l'utilisateur symfony
RUN mkdir /home/symfony/.ssh && \
    chmod 700 /home/symfony/.ssh && \
    chown symfony:symfony /home/symfony/.ssh
```

```
# Changer l'utilisateur
USER symfony

# Installer les dépendances Node.js
RUN npm install
RUN npm run build

# Installer les dépendances PHP
RUN composer install

# Exposer les ports 8080 pour Symfony et 22 pour SSH
EXPOSE 8080 22

# Configurer le shell de l'utilisateur symfony pour qu'il démarre dans
/var/www/symfony
RUN echo "cd /var/www/symfony" >> /home/symfony/.bashrc

# Revenir à l'utilisateur root pour démarrer les services
USER root

# Créer un script de démarrage
RUN echo '#!/bin/bash\n\
service ssh start\n\
symfony server:start --port=8080 --no-tls' > /start.sh && \
chmod +x /start.sh

# Commande par défaut pour démarrer le serveur SSH et Symfony
CMD ["/start.sh"]
```

Figure 3: Dockerfile de l'environnement de développement

## Cas d'utilisation:

L'administrateur hérite des droits utilisateurs, pour chaque action, ils doivent impérativement se connecter, sauf pour réinitialiser leur mot de passe.

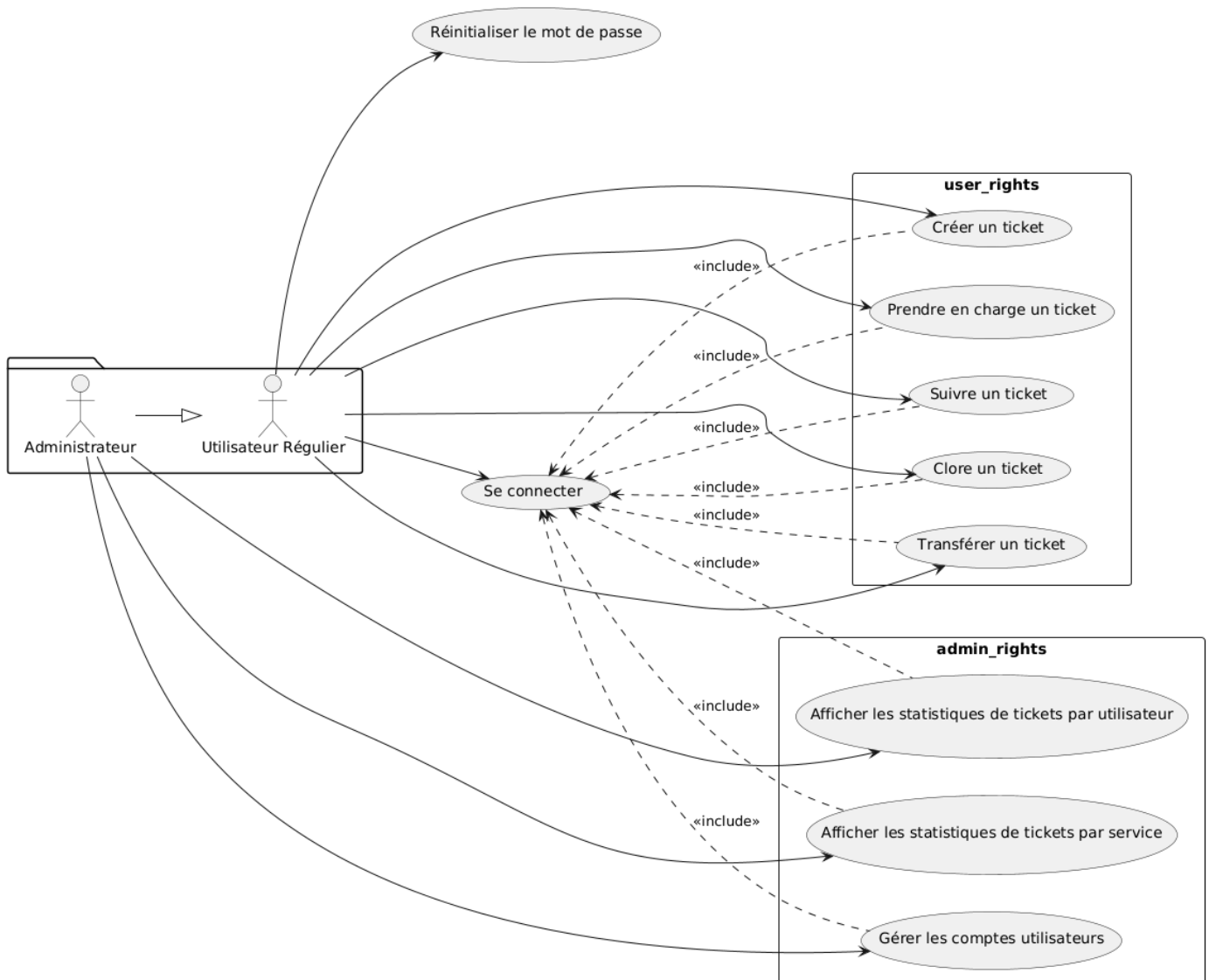


Figure 4: Diagramme de cas d'utilisation du projet "PLATEFORME TICKETING"

## V. Réalisations permettant la mise en œuvre des compétences

### Conception des maquettes:

Les maquettes ont été conçues avec l'outil Figma.

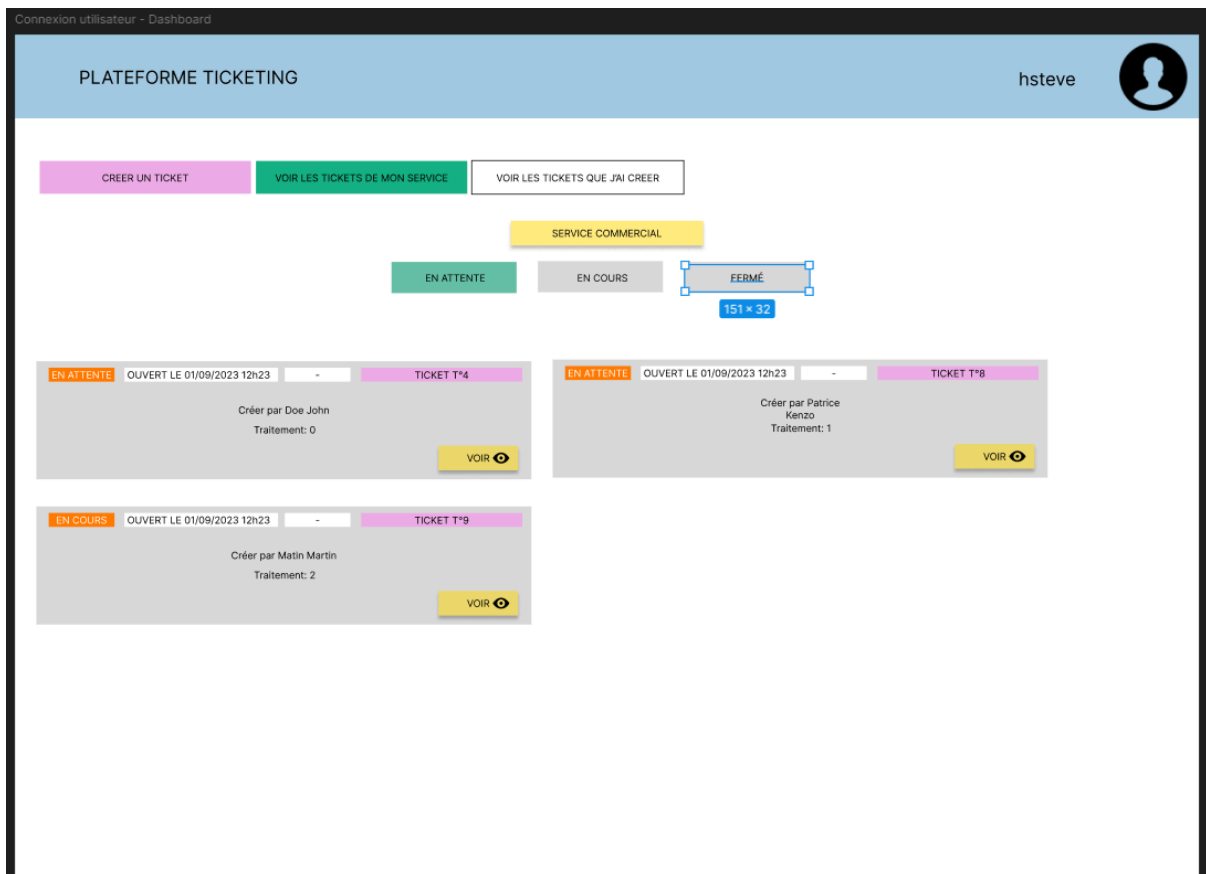


Figure 5: Exemple de maquette conçue pour l'application, page d'accueil pour un utilisateur

## Enchaînement de maquette des interfaces:

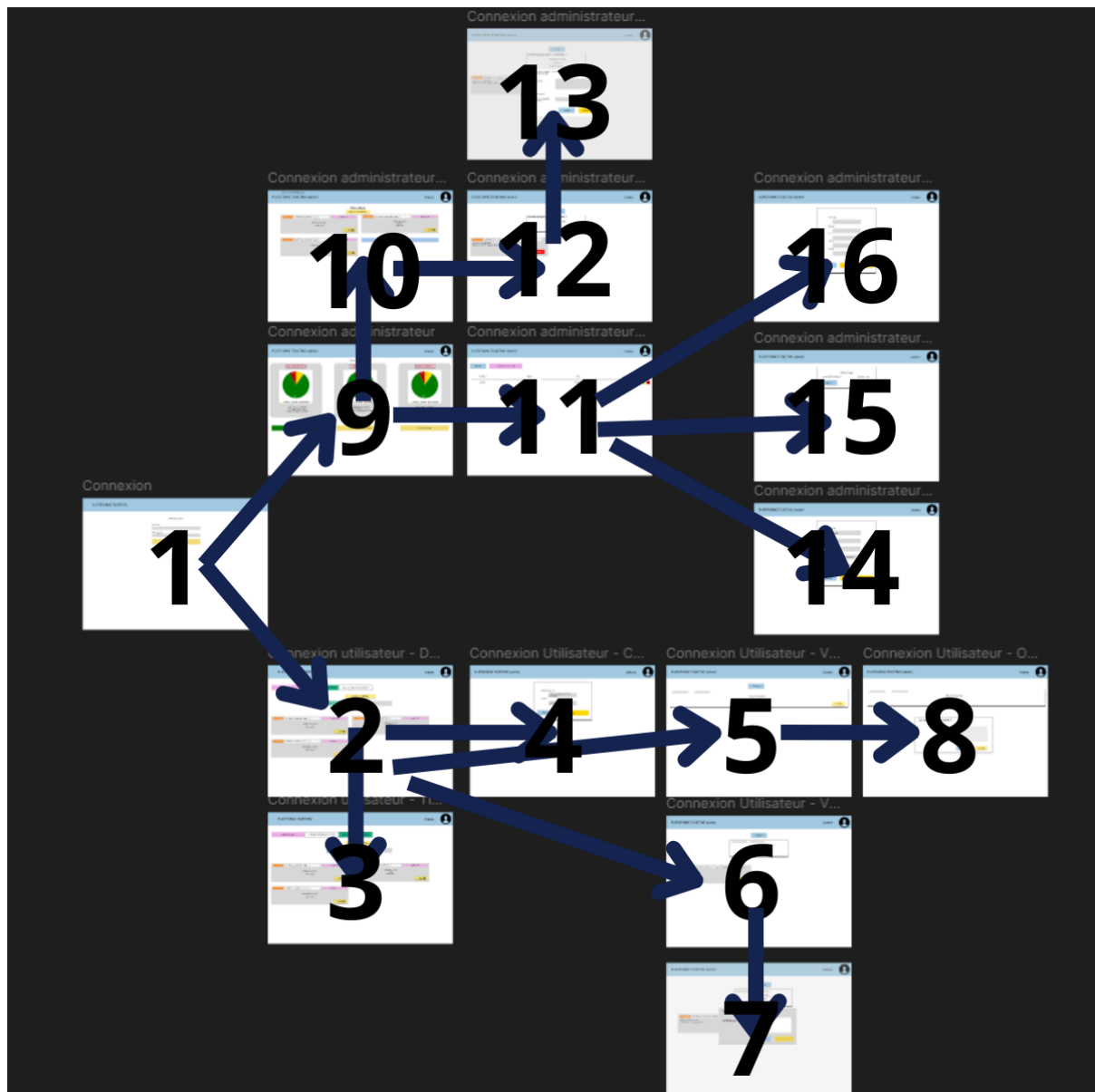


Figure 6: Enchaînement de maquette des interfaces

Page 1: Interface de connexion (commune à tous)

Interfaces utilisateur:

- Page 2: Page d'accueil, accès à tous les tickets du service
- Page 3: Page d'accueil, accès à tous les tickets que l'utilisateur a créé
- Page 4: Page de création d'un ticket
- Page 5: Page de visualisation d'un ticket
- Page 6: Page de visualisation des traitement d'un ticket
- Page 7: Page de fermeture d'un ticket
- Page 8: Page d'ouverture d'un ticket

Interfaces administrateur:

- Page 9: Page d'accueil, tableau de bord

- Page 10: Page historique des tickets par service
- Page 11: Page liste des utilisateurs
- Page 12: Page de visualisation de ticket
- Page 13: Page de relance d'un ticket
- Page 14: Page de modification des informations d'un utilisateur
- Page 15: Page de visualisation des informations d'un utilisateur
- Page 16: Page de création d'un utilisateur

## Modèle conceptuel des données:

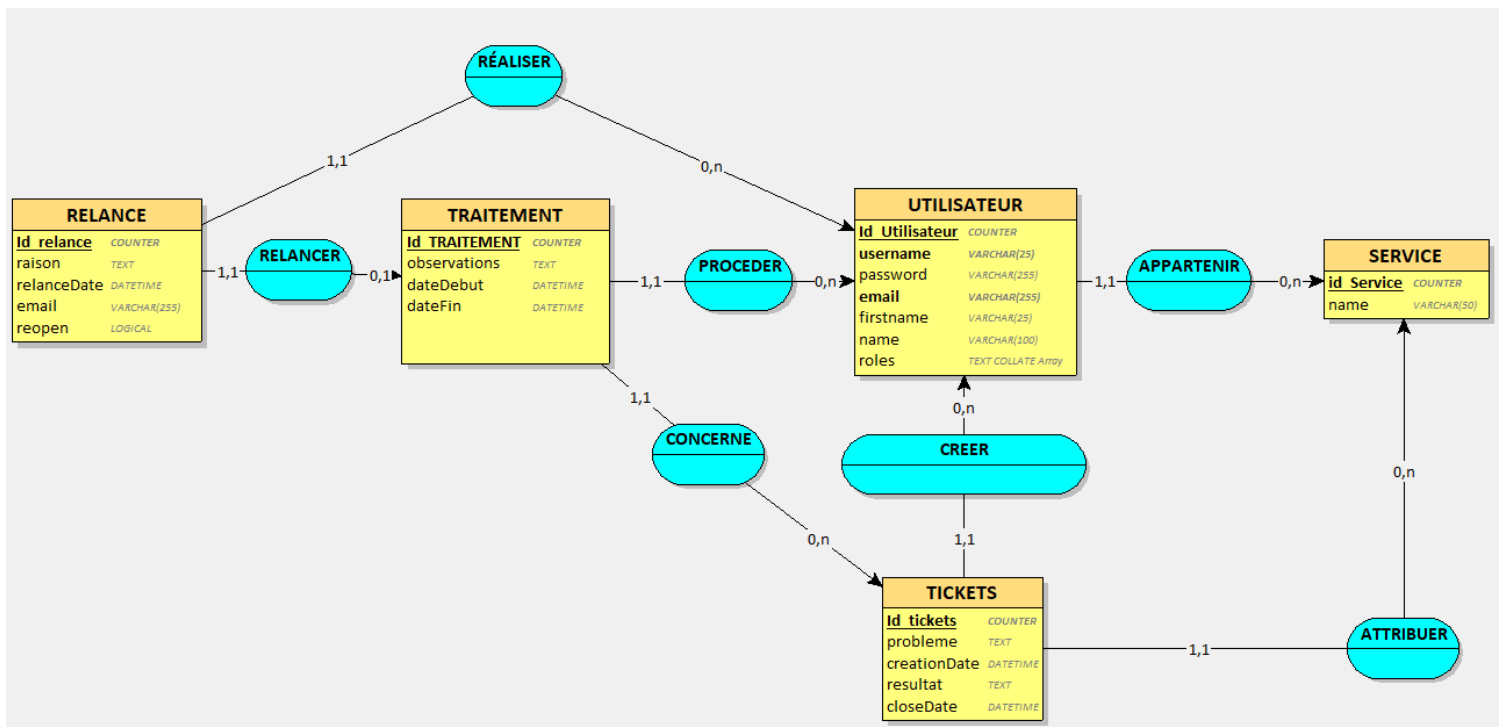


Figure 7: Modèle Conceptuel des données (ou MCD)

Dans ce MCD, qui est un modèle entité-association, 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:

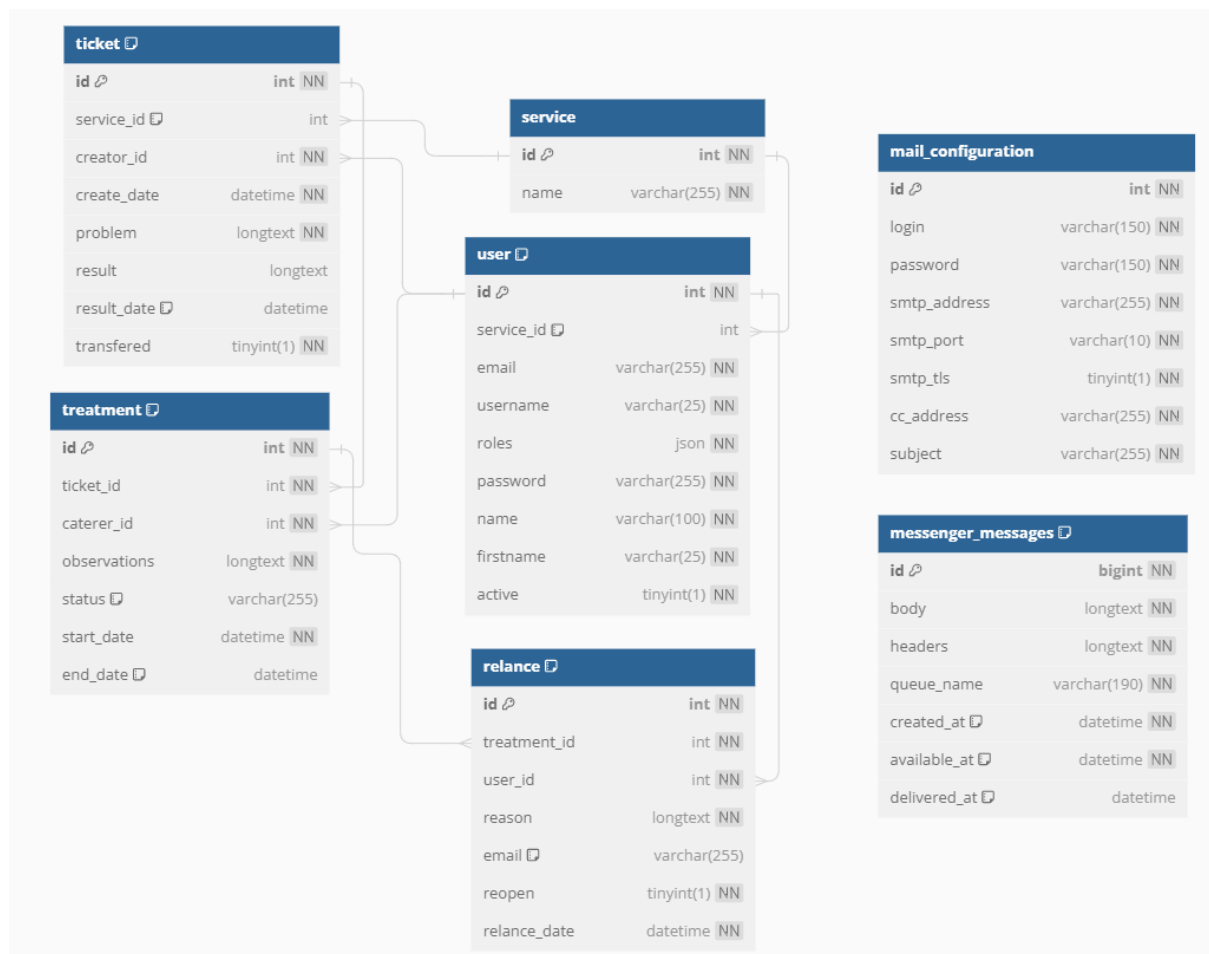


Figure 8: Modèle physique des données

Le modèle conceptuel des données a été transformé en un modèle physique composé de 7 tables dans la base de données MySQL. Les 5 entités initiales ont été directement converties en tables, et aucune association n'a été transformée en table.

Voici les correspondances entre les entités du modèle conceptuel et les tables du modèle physique :

1. L'entité **Service** est devenue la table **service**, avec une clé primaire **id**.
2. L'entité **Utilisateur** est devenue la table **user**, avec une clé primaire **id** et une clé étrangère **service\_id** qui fait référence à la table **service**.
3. L'entité **Tickets** est devenue la table **ticket**, avec une clé primaire **id**, une clé étrangère **service\_id** pointant vers la table **service** et une clé étrangère **creator\_id** pointant vers la table **user**.
4. L'entité **Traitement** est devenue la table **treatment**, avec une clé primaire **id**, une clé étrangère **ticket\_id** pointant vers la table **ticket**, et une clé étrangère **caterer\_id** pointant vers la table **user**.
5. L'entité **Relance** est devenue la table **relance**, avec une clé primaire **id**, une clé étrangère **treatment\_id** pointant vers la table **treatment**, et une clé étrangère **user\_id** pointant vers la table **user**.

Deux nouvelles tables ont été ajoutées pour répondre à des besoins spécifiques de l'application :

1. **messenger\_messages** : Cette table, avec une clé primaire **id**, est utilisée pour enregistrer les messages des publishers qui seront traités par un système de file d'attente (**queue**). La configuration de ce module est définie dans le fichier d'environnement **.env**.

```
###> 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 ###
```

Figure 9: Configuration de MESSENGER\_TRANSPORT\_DSN dans le fichier .env

2. **mail\_configuration** : Cette table, avec une clé primaire **id**, est utilisée pour stocker la configuration de l'email de la plateforme. Bien qu'une seule configuration soit prévue dans le cadre de l'utilisation actuelle, cette table permet de gérer une évolution potentielle de la plateforme.



## Diagramme de classe:

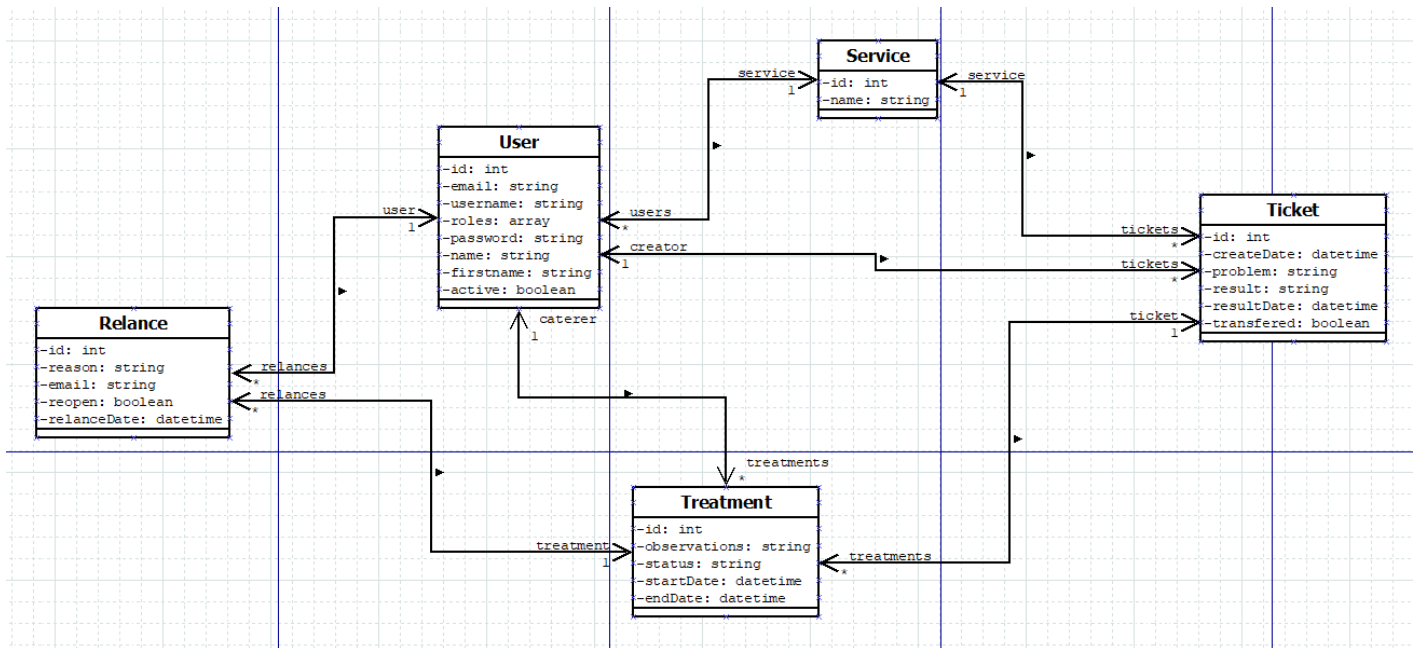


Figure 10: Diagramme de classe de l'application

Le diagramme de classe comporte 5 classes, vu que l'application est développée en Symfony, ces classes sont appelées entités, les méthodes des classes sont donc générées automatiquement.

On distingue 5 entités:

### 1. Relance

- **Attributs :**
  - **id** : Clé primaire (**int**).
  - **reason** : Texte décrivant la raison de la relance (**string**).
  - **email** : Adresse email associée (**string**, nullable).
  - **reopen** : Indicateur de réouverture (**bool**).
  - **relanceDate** : Date de la relance (**DateTimeInterface**).
- **Relations :**
  - **treatment** : Relation OneToOne avec **Treatment**.
  - **user** : Relation ManyToOne avec **User**.

### 2. Service

- **Attributs :**
  - **id** : Clé primaire (**int**).
  - **name** : Nom du service (**string**).
- **Relations :**
  - **tickets** : Relation OneToMany avec **Ticket**.
  - **users** : Relation OneToMany avec **User**.

### 3. Ticket

- **Attributs :**
  - **id** : Clé primaire (**int**).
  - **createDate** : Date de création (**DateTimeInterface**).
  - **problem** : Description du problème (**string**).
  - **result** : Résultat du traitement (**string**, nullable).
  - **resultDate** : Date du résultat (**DateTimeInterface**, nullable).
  - **transferred** : Indicateur de transfert (**bool**).
- **Relations :**
  - **service** : Relation ManyToOne avec **Service**.
  - **creator** : Relation ManyToOne avec **User**.
  - **treatments** : Relation OneToMany avec **Treatment**.

### 4. Treatment

- **Attributs :**
  - **id** : Clé primaire (**int**).
  - **observations** : Observations du traitement (**string**).
  - **status** : Statut du traitement (**string**, nullable).
  - **startDate** : Date de début du traitement (**DateTimeInterface**).
  - **endDate** : Date de fin du traitement (**DateTimeInterface**, nullable).
- **Relations :**
  - **ticket** : Relation ManyToOne avec **Ticket**.
  - **caterer** : Relation ManyToOne avec **User**.

### 5. User

- **Attributs :**
  - **id** : Clé primaire (**int**).
  - **email** : Observations du traitement (**string**, unique).
  - **username** : Statut du traitement (**string**, unique).
  - **password** : Date de début du traitement (**DateTimeInterface**).
  - **roles** : Date de fin du traitement (**Array**).
  - **name** : Date de fin du traitement (**String**).
  - **firstName** : Date de fin du traitement (**String**).
  - **active** : Si l'utilisateur est activé (**Boolean**).
- **Relations :**
  - **service** : Relation ManyToOne avec **Service**
  - **treatments** : Relation OneToMany avec **Treatment**
  - **tickets** : Relation OneToMany avec **Ticket**.

## Conception et développement de la connexion utilisateur:

La connexion utilisateur permet à la plateforme d'être sécurisée, elle va permettre aussi d'identifier le type d'utilisateur et de le rediriger vers les ressources dont il a la permission d'accéder.

La conception a été faite avec l'outil figma (C.f. Annexe 2), l'utilisation de la route "app\_login" permet d'authentifier l'utilisateur. Son mot de passe est hashé en base de données.

La route "app\_logout" accessible avec l'url "/logout" permet de déconnecter l'utilisateur.

## Conception et développement des fonctionnalités pour les utilisateurs:

Pour un utilisateur régulier, il est lié à un service, une fois connecté, il a accès aux tickets de son service.

Il peut créer un ticket (C.f Annexe 3) destiné à un autre service, ouvrir un ticket de son service/créer un traitement d'un ticket non-ouvert, prendre le relais d'un ticket déjà ouvert et clore un ticket. Lorsqu'un traitement est créé, un déclencheur est exécuté en base de données qui permet de mettre à jour le ticket. (C.f Annexe 6). Les champs "Raison d'ouverture" et "Service" sont obligatoires. L'utilisateur appuie sur le bouton "Créer un ticket", qui fait appel à l'url "/ticket/new" donc à la route "app\_ticket\_new".

```
#[Route('/ticket')]
class TicketController extends AbstractController
{
    /**
     * Handles the creation of a new ticket.
     * @throws Exception
     */
    #[Route('/new', name: 'app_ticket_new')]
    public function new(Request $request, EntityManagerInterface $registry):
    Response
    {
        set_time_limit(0);
        if (!$this->getUser()) $this->redirectToRoute("app_login");
        $user = $registry->getRepository(User::class)->findOneBy(['username' =>
$this->getUser()->getUserIdentifier()]);

        $ticket = new Ticket();
        $form = $this->createForm(TicketType::class, $ticket);
        $form->handleRequest($request);
        if ($form->isSubmitted() && $form->isValid()) {
            $ticket->setCreator($user);
            $ticket->setTransferred(false);
            $service =
$this->getRegistry->getRepository(Service::class)->find($form->get('service')->getData());
            if ($service == null) {
                $this->addFlash('fail', "Ce service n'existe pas...");
            } else {
                $ticket->setService($service);
                $ticket->setCreateDate(new DateTime("now", new
DateTimeZone($_ENV["DATETIMEZONE"])));
                $registry->persist($ticket);
                $registry->flush();
                $this->addFlash('success', "Vous avez ouvert un nouveau ticket pour
le service " . $service->getName());
                return $this->redirectToRoute("app_main");
            }
        }
        return $this->render("ticket/new.html.twig", [
            'form' => $form,
            'serviceLst' => $registry->getRepository(Service::class)->findAll()
        ]);
    }
}
```

Figure 11: Code de la route “app ticket new”

## Jeu d'essai:

Données attendues	Type
Raison d'ouverture	Chaîne de caractères allant de 10 à 500 caractères
Service	Entier positif

Données en entrées	Données/Erreurs obtenues
<ul style="list-style-type: none"><li>- Une raison d'ouverture allant de 10 à 500 caractères</li><li>- L'identifiant d'un service existant en entier positif</li></ul>	<div>Vous avez ouvert un nouveau ticket pour le service</div>
<ul style="list-style-type: none"><li>- Une raison d'ouverture en dessous de 10 caractères</li></ul>	<div>La raison doit contenir au moins 10 caractères.</div>
<ul style="list-style-type: none"><li>- Un entier négatif ou un identifiant de service non-existant</li></ul>	<div>Ce service n'existe pas...</div>

Des contraintes permettent de valider les données en entrée:

```
class TicketType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('problem', TextareaType::class, [
                'label' => "Raison de l'ouverture du ticket",
                'label_attr' => ['class' => 'label'],
                'attr' => ['class' => 'input input-bordered', 'placeholder' => 'Le
client CL001 a besoin de...'],
                "mapped" => true,
                "required" => true,
                "trim" => true,
                'constraints' => [
                    new NotBlank([
                        'message' => 'La raison de l\'ouverture du ticket ne peut
pas être vide.',
                    ]),
                    new Length([
                        'min' => 10,
                        'minMessage' => 'La raison doit contenir au moins {{ limit
}} caractères.',
                        'max' => 500,
                        'maxMessage' => 'La raison ne peut pas dépasser {{ limit }}
caractères.',
                    ]),
                ],
            ],
        );
        $builder
            ->add('service', TextType::class, [
                'label' => "Service",
                'label_attr' => ['class' => 'label'],
                'attr' => ['class' => 'input input-bordered', 'placeholder' =>
'Rechercher...', 'list' => 'serviceLst'],
                "mapped" => false,
                "required" => true,
                "trim" => true,
            ],
        );
    }

    public function configureOptions(OptionsResolver $resolver): void
    {
        $resolver->setDefaults([
            'data_class' => Ticket::class,
        ]);
    }
}
```

Figure 12: Code du formulaire de ticket

## Tests unitaires:

Dans la classe “TicketControllerTest” on retrouve différentes fonctions de test, permettant de valider le développement.

On affirme que l'utilisateur est créé (C.f. Annexe 5) et qu'après la création de son ticket, il est redirigé vers la page d'accueil et que son ticket est bien mis en attente.

```
public function testCreateTicket()
{
    $user = $this->createUserAndLogin();
    $service = $this->createService();
    $user->setService($service);
    $this->entityManager->flush();

    // Request the ticket creation page
    $crawler = $this->client->request('GET', '/ticket/new');
    $this->assertResponseIsSuccessful();

    // Submit the form to create a new ticket
    $form = $crawler->selectButton('Créer')->form([
        'ticket[problem]' => 'Problème de test',
        'ticket[service]' => $service->getId(),
    ]);

    $this->client->submit($form);

    // Assert redirection and verify the ticket was created
    $this->assertResponseRedirects('/');
    $this->client->followRedirect();

    $ticket =
    $this->entityManager->getRepository(Ticket::class)->findOneBy(['problem' =>
    'Problème de test']);
    $this->assertNotNull($ticket);
    $this->assertEquals($user->getId(), $ticket->getCreator()->getId());
    $this->assertEquals($service->getId(), $ticket->getService()->getId());
    $this->assertFalse($ticket->isTransferred());
    $this->assertEquals("EN ATTENTE", $ticket->getStatus());
}
```

Figure 13: Fonction “testCreateTicket()” de la classe “TicketControllerTest”

## Visuel final:

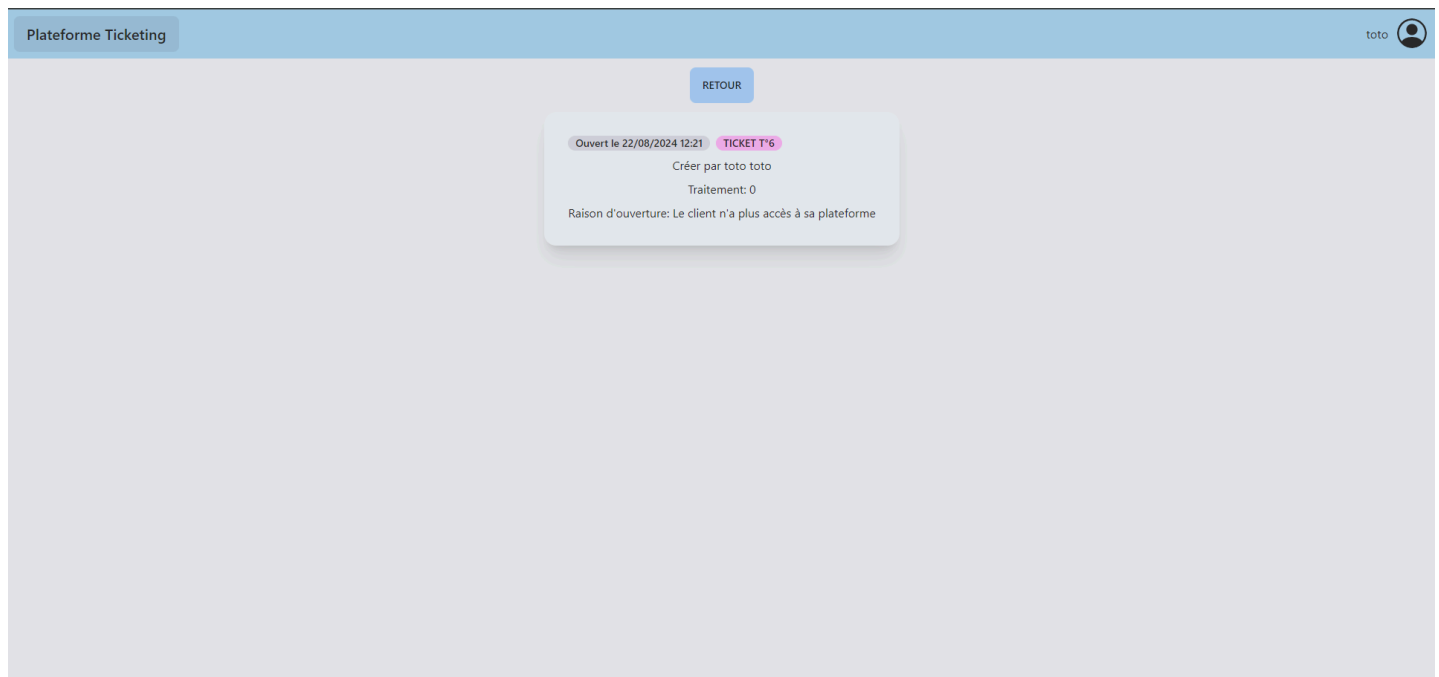


Figure 14: Interface final de visualisation d'un ticket créé par l'utilisateur "Toto"

## Conception et développement des fonctionnalités pour les administrateurs:

Pour un administrateur, il hérite des propriétés de l'utilisateur mais on le distingue par un rôle ("ROLE\_ADMIN"), une fois connecté, il a accès aux tickets de chaque service par une représentation en tableau de bord. (C.f Annexe 7).

L'administrateur peut faire la gestion des comptes, donc créer, modifier ou supprimer un utilisateur.

L'administrateur a un accès aux historiques de traitement pour chaque ticket de service et des traitements par utilisateurs.

Il peut ainsi voir le nombre de tickets par statut par service et par utilisateurs.



```
{% if "ROLE_ADMIN" in app.user.roles %}
    <div>
        {% if dashboard == true %}
            <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 %}
                    <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'information #}
                            <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: 20%">
                                <canvas id="canvaService{{ service.id }}"></canvas>
                            </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 qui se met à jour après le chargement du contenu du DOM #}
                                <span id="inProgressBadgeIdService{{ service.id }}"
                                    class="badge badge-info">-</span>
                                <span id="closeBadgeIdService{{ service.id }}"
                                    class="badge badge-success">-</span>
                            </div>
                        </a>
                    </div>
                {% endfor %}
            </div>
        {% else %}
            {{ include('ticket/see.one_service_tickets.html.twig') }}
        {% endif %}
    </div>
    <div class="mt-3 ml-2 mb-3 grid grid-cols-3 gap-4">
        {% if dashboard %}
            <a href="{{ path('app_main') }}"?dashboard=false" class="btn btn-success
w-full">VOIR LES TICKETS DE
                MON
                SERVICE</a>
        {% else %}
            <a href="{{ path('app_main') }}" class="btn btn-success w-full">VOIR LE
DASHBOARD</a>
        {% endif %}
        <a href="{{ path('app_user') }}" class="btn btn-primary w-full">GESTION DES
UTILISATEURS</a>
        <a href="{{ path('app_configuration_mail') }}" class="btn btn-primary
w-full">PARAMETRE MAIL</a>
    </div>
{% else %}{# L'utilisateur n'est pas ADMIN -> Mode user #}
    {{ include('ticket/see.one_service_tickets.html.twig') }}
{% endif %}
```

Figure 15: Code de la vue "index.html.twig"

Une requête AJAX (C.f. Annexe 8) sécurisée par un CSRF Token, demande le nombre de tickets à afficher dans les graphiques.

La route “app\_api\_count\_tickets\_service” est donc appelée.

```
/**
 * Counts and returns the number of tickets for a specific service.
 *
 * This POST-only endpoint uses a stored procedure to get counts for tickets in
 * waiting, in progress, and closed states.
 * It validates the CSRF token before executing the procedure.
 *
 * @param Request $request The HTTP request with CSRF token.
 * @param EntityManagerInterface $manager The entity manager.
 * @param int $service The service ID.
 * @return JsonResponse JSON response with ticket counts.
 * @throws Exception If an error occurs.
 */
#[Route('/api/count-tickets/{service}/', name: 'app_api_count_tickets_service',
methods: ['POST'])] //on tolère uniquement les requêtes POST
public function apiCount(Request $request, EntityManagerInterface $manager,
$service): JsonResponse
{
    set_time_limit(0);
    if ($request->request->has('_csrf_token') && $this->isCsrfTokenValid('api-count'
. $service, $request->request->get('_csrf_token'))) { //On vérifie la validité du
jeton
        $conn = $manager->getConnection();
        $stmt = $conn->prepare('CALL count_tickets_service(:serviceId)');
        $result = $stmt->executeQuery(['serviceId' =>
$service])->fetchAssociative();
        return new JsonResponse([
            $result['in_waiting'],
            $result['in_progress'],
            $result['closed']]
        );
    }
    return new JsonResponse(array(500, "CSRF invalid"));
}
```

Figure 16: Code de la route “app\_api\_count\_tickets\_service”

Cette route appelle la procédure “count\_tickets\_service” (C.f. Annexe 9) dans la base de données.

## Jeu d'essai:

Données attendues	Type
Service	Entier

Données en entrées	Données/Erreurs obtenues												
<ul style="list-style-type: none"><li>- L'identifiant d'un service existant en entier positif</li></ul>	<p>Exemple avec le service 1:</p> <table><tr><th colspan="4">Résultat #1 (1r × 3c)</th></tr><tr><th>#</th><th>in_waiting</th><th>in_progress</th><th>closed</th></tr><tr><td>1</td><td>9</td><td>0</td><td>1</td></tr></table>	Résultat #1 (1r × 3c)				#	in_waiting	in_progress	closed	1	9	0	1
Résultat #1 (1r × 3c)													
#	in_waiting	in_progress	closed										
1	9	0	1										
<ul style="list-style-type: none"><li>- Un entier négatif ou un identifiant de service non-existant</li></ul>	<table><tr><th colspan="4">Résultat #1 (1r × 3c)</th></tr><tr><th>#</th><th>in_waiting</th><th>in_progress</th><th>closed</th></tr><tr><td>1</td><td>0</td><td>0</td><td>0</td></tr></table>	Résultat #1 (1r × 3c)				#	in_waiting	in_progress	closed	1	0	0	0
Résultat #1 (1r × 3c)													
#	in_waiting	in_progress	closed										
1	0	0	0										

## Tests d'intégrations:

Symfony utilise Doctrine, qui permet de mettre à jour les bases de données, il y a donc eu une migration qui a été effectuée.

```
final class Version20240817113102 extends AbstractMigration
{
    public function getDescription(): string
    {
        return '';
    }

    public function up(Schema $schema): void
    {
        $this->addSql("DELETE FROM doctrine_migration_versions WHERE version LIKE '%Version20240817113102'");
        $this->addSql('
DROP PROCEDURE IF EXISTS count_tickets_service;
');
        $this->addSql("
CREATE PROCEDURE IF NOT EXISTS count_tickets_service(IN serviceId INT)
BEGIN
SELECT
COUNT(CASE WHEN t.id NOT IN (SELECT DISTINCT ticket_id FROM treatment)
OR latest_treatment.status = 'EN ATTENTE' THEN 1 END) AS in_waiting,
COUNT(CASE WHEN latest_treatment.status = 'EN COURS' THEN 1 END) AS
in_progress,
COUNT(CASE WHEN latest_treatment.status = 'Fermé' THEN 1 END) AS closed
FROM ticket t
LEFT JOIN (
SELECT ticket_id, status
FROM treatment t1
WHERE t1.end_date = (
SELECT MAX(t2.end_date)
FROM treatment t2
WHERE t2.ticket_id = t1.ticket_id
)
) latest_treatment ON t.id = latest_treatment.ticket_id
WHERE t.service_id = serviceId;
END
");
```

Figure 17: Fichier de migration pour intégrer la procédure

Lors d'un déploiement, la procédure sera effacée puis recréée sur le serveur de production.

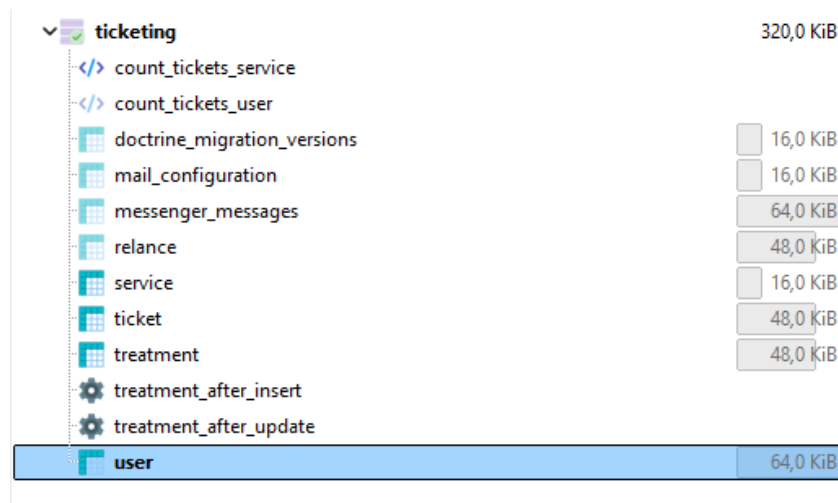


Figure 18: Base de données "ticketing" sur le serveur de production.

Visuel Final:



Figure 19: Interface final de visualisation du tableau de bord pour un utilisateur administrateur

## Affichage des statistiques:

Lorsque l'administrateur voudra voir les statistiques, il devra se rendre dans la gestion des utilisateurs et cliquer sur le bouton "Voir les statistiques"

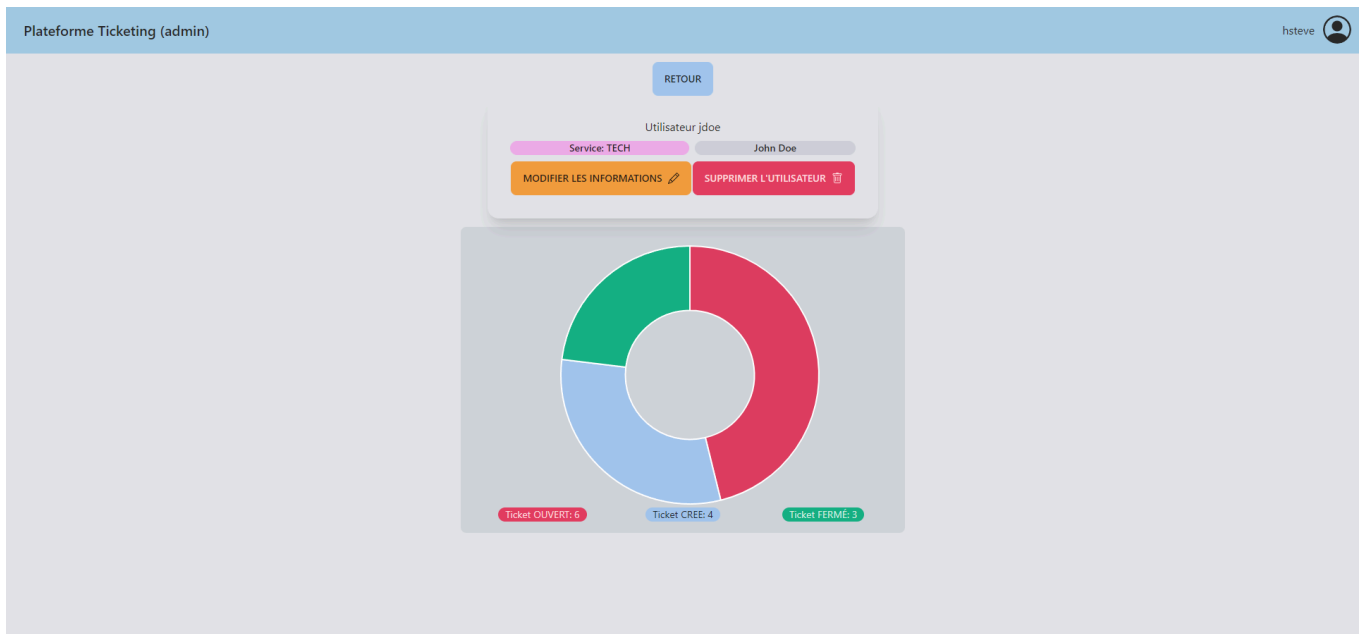


Figure 20: Interface finale administrateur, vision des statistiques de l'utilisateur "jdoe"

Ces statistiques sont chargées par une requête AJAX.

```
$.ajax({
  type: 'POST',
  url: '/api/count-tickets-user/{ user.id }/',
  data: { _csrf_token: "{{ csrf_token('api-count-user'~user.id) }}" },
  success: function (response) {
    if (response !== 500) {
      ms["{{ user.id }}"].data.datasets[0].data[0] = parseFloat(response[0]);
      ms["{{ user.id }}"].data.datasets[0].data[1] = parseFloat(response[1]);
      ms["{{ user.id }}"].data.datasets[0].data[2] = parseFloat(response[2]);
      ms["{{ user.id }}"].update();
      $('#openBadgeIdService{{ user.id }}').html("Ticket OUVERT: " +
response[0]);
      $('#createBadgeIdService{{ user.id }}').html("Ticket CREE: " +
response[1]);
      $('#closeBadgeIdService{{ user.id }}').html("Ticket FERMÉ: " +
response[2]);
    } else {
      console.error('Erreur lors de la requête. (500)');
    }
  },
  error: function (error) {
    console.error('Erreur lors de la requête. ' + error);
  }
});
```

Figure 21: Requête AJAX pour l'affiche des statistiques d'un utilisateur

Cette requête pointe vers la route “app\_api\_count\_tickets\_user”

```
/**
 * Count tickets associated with a specific user via a stored procedure.
 *
 * @param Request $request HTTP request containing user data.
 * @param EntityManagerInterface $manager Manages database interactions.
 * @param int|string $user The user ID to count tickets for.
 * @return JsonResponse JSON response containing ticket counts.
 * @throws Exception Throws exception on database or CSRF issues.
 */
#[Route('/api/count-tickets-user/{user}/', name: 'app_api_count_tickets_user',
methods: ['POST'])]
public function apiUser(Request $request, EntityManagerInterface $manager, $user):
JsonResponse
{
    set_time_limit(0);
    if ($request->request->has('_csrf_token') &&
$this->isCsrfTokenValid('api-count-user' . $user,
$request->request->get('_csrf_token'))) {
        $conn = $manager->getConnection();
        $stmt = $conn->prepare('CALL count_tickets_user(:userId)');
        $result = $stmt->executeQuery(['userId' => $user])->fetchAssociative();
        return new JsonResponse([
            $result['n_open'],
            $result['n_create'],
            $result['n_close']
        ]);
    }
    return new JsonResponse(array(500, "CSRF invalid"));
}
```

Figure 22: Route “app\_api\_count\_tickets\_user”

Cette route fait appelle à la procédure “count\_tickets\_user” stockée en base de données (C.f. Annexe 11).

```
DELIMITER //
CREATE DEFINER=`ticketing`@`%` PROCEDURE `count_tickets_user`(
    IN `userId` INT
)
LANGUAGE SQL
NOT DETERMINISTIC
CONTAINS SQL
SQL SECURITY DEFINER
COMMENT ''
BEGIN
SELECT
    (SELECT COUNT(*) FROM treatment WHERE caterer_id = userId) AS n_open,
    (SELECT COUNT(*) FROM ticket WHERE creator_id = userId) AS n_create,
    (SELECT COUNT(*) FROM treatment t JOIN ticket ti ON t.ticket_id = ti.id WHERE
t.caterer_id = userId AND t.end_date = ti.result_date AND t.`status` = "Fermé") AS
n_close;
END //
```

Figure 23 : Procédure “count\_tickets\_user”

Jeu d'essai:

Données attendues	Type
Utilisateur	Entier

Données en entrées	Données/Erreurs obtenues												
<ul style="list-style-type: none"><li>- L'identifiant d'un utilisateur existant en entier positif</li></ul>	<p>Exemple avec le l'utilisateur 3:</p> <table><tr><th colspan="4">Résultat #1 (1r × 3c)</th></tr><tr><th>#</th><th>n_open</th><th>n_create</th><th>n_close</th></tr><tr><td>1</td><td>6</td><td>4</td><td>3</td></tr></table>	Résultat #1 (1r × 3c)				#	n_open	n_create	n_close	1	6	4	3
Résultat #1 (1r × 3c)													
#	n_open	n_create	n_close										
1	6	4	3										
<ul style="list-style-type: none"><li>- Un entier négatif ou un identifiant d'utilisateur non-existant</li></ul>	<table><tr><th colspan="4">Résultat #1 (1r × 3c)</th></tr><tr><th>#</th><th>n_open</th><th>n_create</th><th>n_close</th></tr><tr><td>1</td><td>0</td><td>0</td><td>0</td></tr></table>	Résultat #1 (1r × 3c)				#	n_open	n_create	n_close	1	0	0	0
Résultat #1 (1r × 3c)													
#	n_open	n_create	n_close										
1	0	0	0										

## Plan de tests:

L'intégralité du plan de tests (C.f Annexe 4) permet de savoir ce qu'il y a tester fonctionnellement dans l'application, puis permet d'ajouter des tests unitaires dans l'application. Les tests unitaires sont utilisés pour le contrôleur de ticket et le contrôleur historique, en testant chaque route, ces tests sont exécutés sur GitHub via le GitHub Actions à chaque push sur un pull request qui est traité avec un Workflow (C.f Annexe 10).

Le code reviewer déterminera à l'issue des tests si le développement peut être fusionné sur master.



## Mise en production dans une démarche DevOps

Le développement de l'application se fait en créant de nouvelles branches en partant depuis master et en pointant vers master.

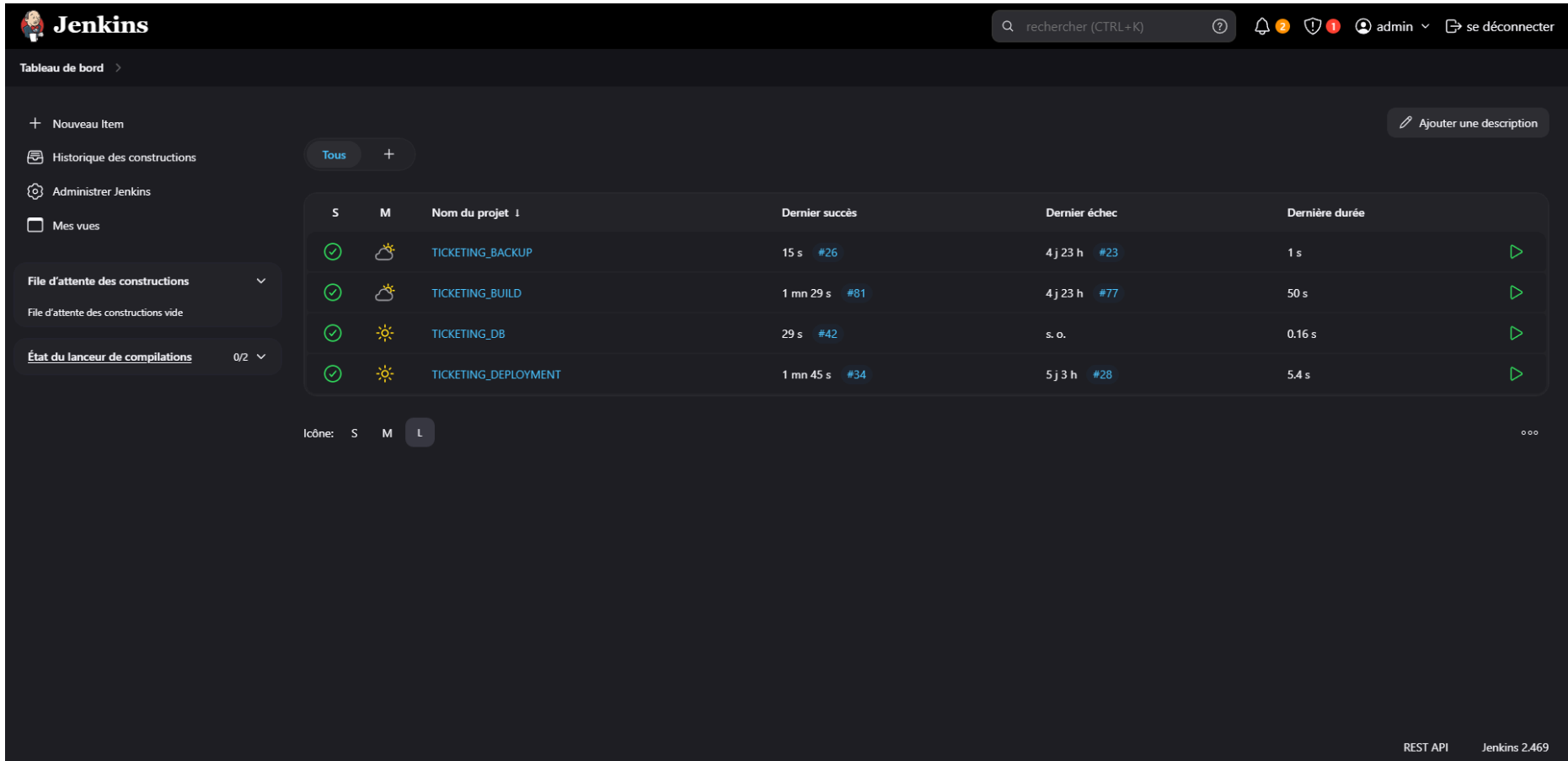
Les branches peuvent être nommées comme:

- feature/\*
- bug/\*
- style/\*
- reformat/\*

Les commits sont nommés comme:

- [feature]
- [refactor]
- [bug]
- [style]
- [reformat]

Une fois la pull request fusionnée avec master, Jenkins récupère le git puis installe l'application en production.



**Jenkins**

rechercher (CTRL+K) ? 2 1 admin se déconnecter

Tableau de bord

+ Nouveau Item

Historique des constructions

Administrer Jenkins

Mes vues

File d'attente des constructions 0/2

File d'attente des constructions vide

État du lanceur de compilations 0/2

Ajouter une description

S	M	Nom du projet	Dernier succès	Dernier échec	Dernière durée
✓	☀	TICKETING_BACKUP	15 s #26	4 j 23 h #23	1 s
✓	☀	TICKETING_BUILD	1 mn 29 s #81	4 j 23 h #77	50 s
✓	☀	TICKETING_DB	29 s #42	s. o.	0.16 s
✓	☀	TICKETING_DEPLOYMENT	1 mn 45 s #34	5 j 3 h #28	5.4 s

Icône: S M L

REST API Jenkins 2.469

Figure 24: Jenkins sur le serveur de production

## La connexion HTTPS garantit la sécurité.

```
<VirtualHost *:80>
    ServerAdmin webmaster@localhost

    # Redirection de tout le trafic vers HTTPS
    Redirect permanent / https://192.42.8.92/

    # Configuration des logs
    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
    LogLevel warn
</VirtualHost>
<VirtualHost *:443>
    ServerAdmin webmaster@localhost
    DocumentRoot /var/jenkins_home/workspace/TICKETING_DEPLOYMENT/web/public

    # Configuration SSL
    SSLEngine on
    SSLCertificateFile /etc/ssl/certs/certificat.crt
    SSLCertificateKeyFile /etc/ssl/private/cle.key
    # Si vous avez un certificat intermédiaire :
    # SSLCertificateChainFile /chemin/vers/votre/chaine.crt

    # Configuration du répertoire racine
    <Directory /var/jenkins_home/workspace/TICKETING_DEPLOYMENT/web/public>
        Options Indexes FollowSymLinks
        AllowOverride All
        Require all granted
        <Files ".ht*">
            Require all denied
        </Files>
    </Directory>

    # Configuration des logs
    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
    LogLevel warn

    # Activer le module rewrite si nécessaire
    <IfModule mod_rewrite.c>
        RewriteEngine On
    </IfModule>

    # Paramètres de sécurité supplémentaires
    # Désactiver la signature du serveur
    ServerSignature Off
</VirtualHost>
```

Figure 24: Configuration d'Apache2

## Description de la veille, effectuée par le candidat durant le projet, sur les vulnérabilités de sécurité, description des vulnérabilités éventuellement trouvées et des failles potentiellement corrigées:

Au cours du développement du projet, j'ai effectué des recherches sur les documentations de Symfony afin d'utiliser les migrations de Doctrine.

J'ai aussi utilisé Stackoverflow pour toutes autres types de recherches.

Lorsque j'ai repris mon projet de formation, j'ai dû exécuter la commande "composer install", celle-ci exécute la commande "composer audit", qui permet d'avoir un audit de sécurité par rapport aux failles présentes dans mon projet.

```
C:\Users\hoare\Desktop\PROJET_FINAL\Ticketing\web>composer audit
The new audit.abandoned setting (currently defaulting to "report" will default to "fail" in Composer 2.7, make sure to set it to "report" or "ignore" explicitly by then if you do not want this.
Found 2 security vulnerability advisories affecting 2 packages:

Package      symfony/security-http
CVE           CVE-2023-46733
Title        CVE-2023-46733: Possible session fixation
URL           https://symfony.com/cve-2023-46733
Affected versions >=5.4.0,<5.4.31|>=6.0.0,<6.1.0|>=6.1.0,<6.2.0|>=6.2.0,<6.3.0|>=6.3.0,<6.3.8
Reported at   2023-11-10T08:00:00+00:00

Package      symfony/twig-bridge
CVE           CVE-2023-46734
Title        CVE-2023-46734: Potential XSS vulnerabilities in CodeExtension filters
URL           https://symfony.com/cve-2023-46734
Affected versions >=2.0.0,<2.1.0|>=2.1.0,<2.2.0|>=2.2.0,<2.3.0|>=2.3.0,<2.4.0|>=2.4.0,<2.5.0|>=2.5.0,<2.6.0|>=2.6.0,<2.7.0|>=2.7.0,<2.8.0|>=2.8.0,<3.0.0|>=3.0.0,<3.1.0|>=3.1.0,<3.2.0|>=3.2.0,<3.3.0|>=3.3.0,<3.4.0|>=3.4.0,<3.5.0|>=3.5.0,<3.6.0|>=3.6.0,<3.7.0|>=3.7.0,<3.8.0|>=3.8.0,<3.9.0|>=3.9.0,<4.0.0|>=4.0.0,<4.1.0|>=4.1.0,<4.2.0|>=4.2.0,<4.3.0|>=4.3.0,<4.4.0|>=4.4.0,<4.5.0|>=4.5.0,<4.6.0|>=4.6.0,<4.7.0|>=4.7.0,<4.8.0|>=4.8.0,<4.9.0|>=4.9.0,<5.0.0|>=5.0.0,<5.1.0|>=5.1.0,<5.2.0|>=5.2.0,<5.3.0|>=5.3.0,<5.4.0|>=5.4.0,<5.4.31|>=5.4.31,<6.0.0|>=6.0.0,<6.1.0|>=6.1.0,<6.2.0|>=6.2.0,<6.3.0|>=6.3.0,<6.3.8
Reported at   2023-11-10T08:00:00+00:00

C:\Users\hoare\Desktop\PROJET_FINAL\Ticketing\web>
```

Figure 25: Audit de sécurité "composer audit"

M'apercevant des failles, j'ai exécuté la commande "composer update" qui a permis de mettre à jour mon application et ainsi de la rendre sécurisée.

```
C:\Users\hoare\Desktop\PROJET_FINAL\Ticketing\web>composer audit
The new audit.abandoned setting (currently defaulting to "report" will default to "fail" in Composer 2.7, make sure to set it to "report" or "ignore" explicitly by then if you do not want this.
No security vulnerability advisories found.

C:\Users\hoare\Desktop\PROJET_FINAL\Ticketing\web>
```

Figure 26: Audit de sécurité "composer audit" après une mise à jour

Une CVE (Common Vulnerabilities and Exposures) du package "symfony/twig-bridge", une potentielle faille XSS(Cross-Site Scripting) était présente.

Je me suis alors demandé si les injections de script peuvent être possibles.

En injectant du script, on peut récupérer le cookie de l'utilisateur administrateur s'il ouvre notre injection, ce qui nous permettra d'accéder à la plateforme avec le compte administrateur.

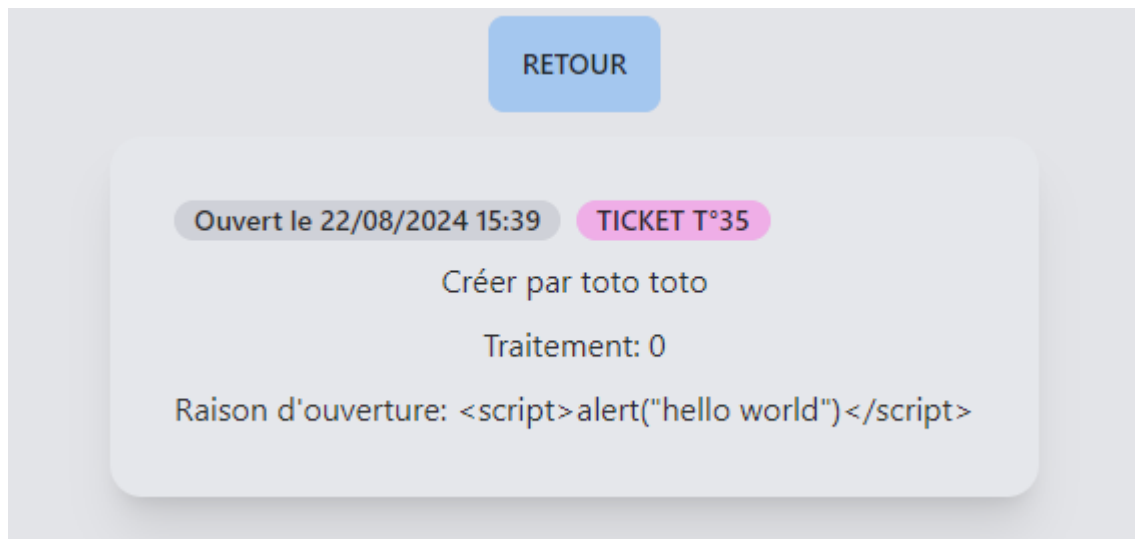


Figure 27: Attaque XSS

Twig échappe les caractères, la plateforme n'est pas vulnérable face aux attaques XSS.

## VI. Annexes

### Annexe 1: Documentation technique de l'environnement de développement:

```
# Plateforme Ticketing

La plateforme de tickets permettant aux services des entreprises de créer et
traiter des tâches par le biais de tickets.

# Logiciels requis

- [DockerDesktop] (https://docs.docker.com/desktop/install/windows-install/)
- [GitScm] (https://git-scm.com/)
- [PhpStorm] (https://www.jetbrains.com/phpstorm/)
- [HeidiSql] (https://www.heidisql.com/download.php)

# Installation du projet

- Récupérer le projet:

```sh
git clone <repo>
```

Remplacer le `<repo>` par l'url du projet GitHub

- Se placer dans le répertoire du projet

- Build l'environnement:

```sh
docker compose build
```

- Démarrer l'environnement

```sh
docker compose up -d
```

(Eteindre l'environnement si nécessaire: `docker compose down`)

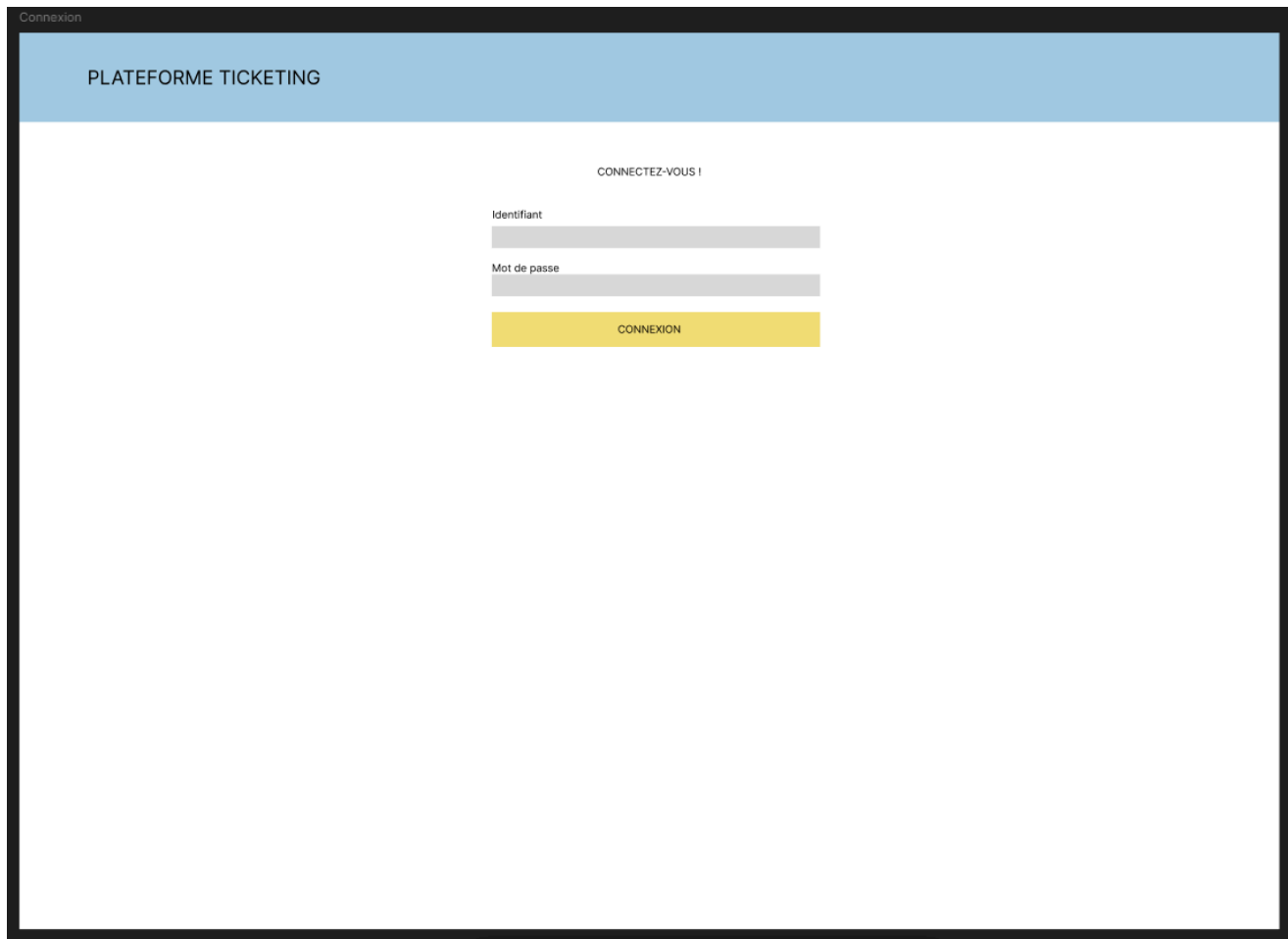
# Commencer le développement

- Ouvrir PhpStorm
- Se connecter en remote développement en utilisant les informations
`symfony@localhost` et le mot de passe `symfony`
- Vérifier que le serveur développement de symfony est ouvert en allant sur
`http://localhost:8080`
- Vérifier que la base de donnée de développement est accessible en se connectant
avec `HeidiSql` et les informations
`ticketing:ticketing@localhost:3306/ticketing`
- Créer ou accéder à votre branche de développement

# Après un développement
```

```
- Test le bon fonctionnement de votre code
- Créer des [Tests Unitaires] (tests.md)
  (Vous pouvez utiliser les informations `root:rootpassword@localhost:3306` pour
  créer une base de donnée de test)
- Créer une version de migration lorsque des entités ont été créées ou modifiées
  `php bin/console make:migration` (et
  n'oubliez pas de les compléter)
- Engager vos modifications
- Pousser le contenu de votre branche sur le répertoire distant GitHub
- Créer une Pull Request allant vers master (attention à toujours être à jour par
  rapport à master et d'ajouter dans la
  description de votre PR l'url de la branche parente)
```

## Annexe 2: Interface utilisateur, page de connexion:




The image shows a user login interface for a ticketing platform. At the top left, the word "Connexion" is written in small text. Below it, a light blue header bar contains the text "PLATEFORME TICKETING". The main content area is white and features a centered login form. The form starts with the instruction "CONNECTEZ-VOUS !". Below this, there are two input fields: the first is labeled "Identifiant" and the second is labeled "Mot de passe". Both fields are represented by light gray rectangular boxes. At the bottom of the form is a yellow rectangular button with the text "CONNEXION" in black capital letters.

Cette interface représente la page 1 de l'enchaînement des maquettes (C.f Annexe 1)



## Annexe 3: Interface utilisateur, création de ticket:

Connexion Utilisateur - Création de tickets

PLATEFORME TICKETING (admin) pkenzo 

Raison d'ouverture

Le client CL 001 a besoin d'un nouveau PC

Service

Commercial

RETOUR CREER

Cette interface représente la page 4 de l'enchaînement des maquettes (C.f Annexe 1)

## Annexe 4: Intégralité du plan de tests:

| ID Test | Nom du Test                             | Conditions préalables  | Étapes du test                             | Résultats attendus   |
|---------|---|--|--|--|
| 1       | Connexion utilisateur régulier          | Aucun  | 1. Accéder à la page de connexion.         | L'utilisateur est redirigé vers la page d'accueil avec le statut 'Connecté'.         |
|         |   | 2. Saisir les identifiants valides d'un utilisateur régulier.                    |  |  |
|         |   | 3. Cliquer sur 'Se connecter'.   |  |  |
| 2       | Connexion administrateur                | Aucun  | 1. Accéder à la page de connexion.         | L'administrateur est redirigé vers le tableau de bord avec le statut 'Connecté'.     |
|         |   | 2. Saisir les identifiants valides d'un administrateur.                          |  |  |
|         |   | 3. Cliquer sur 'Se connecter'.   |  |  |
| 3       | Déconnexion utilisateur                 | L'utilisateur doit être connecté.  | 1. Accéder au menu utilisateur.            | L'utilisateur est redirigé vers la page de connexion avec le statut 'Déconnecté'.    |
|         |   | 2. Cliquer sur 'Se déconnecter'.   |  |  |
| 4       | Réinitialisation du mot de passe        | Aucun  | 1. Accéder à la page de connexion.         | L'utilisateur reçoit un email avec un nouveau mot de passe généré aléatoirement.     |
|         |   | 2. Cliquer sur 'Mot de passe oublié'.  |  |  |
|         |   | 3. Saisir l'adresse email et soumettre.  |  |  |
| 5       | Création de ticket utilisateur régulier | L'utilisateur doit être connecté.  | 1. Accéder à la section 'Créer un ticket'. | Le ticket est créé et affiché dans la liste des tickets avec le statut 'EN ATTENTE'. |
|         |   | 2. Saisir les informations du ticket (raison, service).                          |  |  |
|         |   | 3. Soumettre le formulaire de création.  |  |  |
| 6       | Suivi de ticket utilisateur régulier    | L'utilisateur doit avoir des tickets assignés.                                   | 1. Accéder à la section 'Mes tickets'.     | L'utilisateur peut visualiser les processus et statuts associés à chaque ticket.     |
| 7       | Prise en charge d'un ticket en attente  | L'utilisateur doit être connecté et avoir un ticket avec le statut 'EN ATTENTE'. | 1. Accéder à la section 'Mes tickets'.     | Le ticket passe au statut 'EN COURS' et est assigné à l'utilisateur.                 |

|    |   |   |  |  |
|----|---|---|--|--|
|    |   | 2. Sélectionner un ticket avec le statut 'EN ATTENTE'.        |  |  |
|    |   | 3. Cliquer sur 'Prendre en charge'.                           |  |  |
| 8  | Clôture d'un ticket                                   | L'utilisateur doit être connecté et avoir un ticket en cours. | 1. Accéder à la section 'Mes tickets'.   | Le ticket est marqué comme 'FERMÉ'.  |
|    |   | 2. Sélectionner un ticket avec le statut 'EN COURS'.          |  |  |
|    |   | 3. Cliquer sur 'Clôturer le ticket'.                          |  |  |
| 9  | Transfert d'un ticket                                 | L'utilisateur doit être connecté et avoir un ticket en cours. | 1. Accéder à la section 'Mes tickets'.   | Le ticket est marqué comme 'TRANSFÉRÉ // EN ATTENTE' et assigné au nouveau service.                |
|    |   | 2. Sélectionner un ticket avec le statut 'EN COURS'.          |  |  |
|    |   | 3. Cliquer sur 'Transférer' et choisir un service.            |  |  |
| 10 | Accès au tableau de bord administrateur               | L'administrateur doit être connecté.                          | 1. Accéder à la section 'Tableau de bord'.                                     | L'administrateur visualise les statistiques des états des tickets par service.                     |
| 11 | Gestion des utilisateurs par l'administrateur         | L'administrateur doit être connecté.                          | 1. Accéder à la section 'Gestion des utilisateurs'.                            | L'administrateur peut créer, modifier, ou supprimer des utilisateurs.                              |
|    |   | 2. Créer, modifier ou supprimer un utilisateur.               |  |  |
| 12 | Accès aux statistiques utilisateur par administrateur | L'administrateur doit être connecté.                          | 1. Accéder à la section 'Statistiques utilisateurs'.                           | L'administrateur visualise le nombre de tickets 'Créé', 'Ouvert', 'Fermé' pour chaque utilisateur. |
| 13 | Récupération du nombre de tickets par service         | Aucun   | 1. Exécuter la procédure de récupération du nombre de tickets par service.     | Le nombre de tickets par service est affiché correctement.   |
| 14 | Récupération du nombre de tickets par utilisateur     | Aucun   | 1. Exécuter la procédure de récupération du nombre de tickets par utilisateur. | Le nombre de tickets par utilisateur est affiché correctement.                                     |

## Annexe 5: Fonctions “up()”, “createUserAndLogin()” et “createService()” de la classe “TicketControllerTest”:

```
class TicketControllerTest extends WebTestCase
{
    private $client;
    private $entityManager;
    private $passwordHasher;

    protected function setUp(): void
    {
        $this->client = static::createClient();
        $this->entityManager =
$this->client->getContainer()->get('doctrine')->getManager();
        $this->passwordHasher =
$this->client->getContainer()->get('security.password_hasher');
    }

    private function createUserAndLogin(): User
    {
        $userRepository = $this->entityManager->getRepository(User::class);

        $existingUser = $userRepository->findOneBy(['email' => 'test@example.com']);

        if ($existingUser) {
            $this->client->loginUser($existingUser);
            return $existingUser;
        }

        $user = new User();
        $user->setEmail('test@example.com');
        $user->setUsername('testuser');
        $user->setName('Test');
        $user->setFirstname('User');
        $user->setActive(true);
        $user->setRoles(['ROLE_USER']);

        $hashedPassword = $this->passwordHasher->hashPassword($user, 'testpassword');
        $user->setPassword($hashedPassword);

        $this->entityManager->persist($user);
        $this->entityManager->flush();

        $this->client->loginUser($user);

        return $user;
    }
}
```

```
private function createService(): Service
{
    $service = new Service();
    $service->setName('Test Service');
    // Add other necessary properties for the Service

    $this->entityManager->persist($service);
    $this->entityManager->flush();

    return $service;
}
```

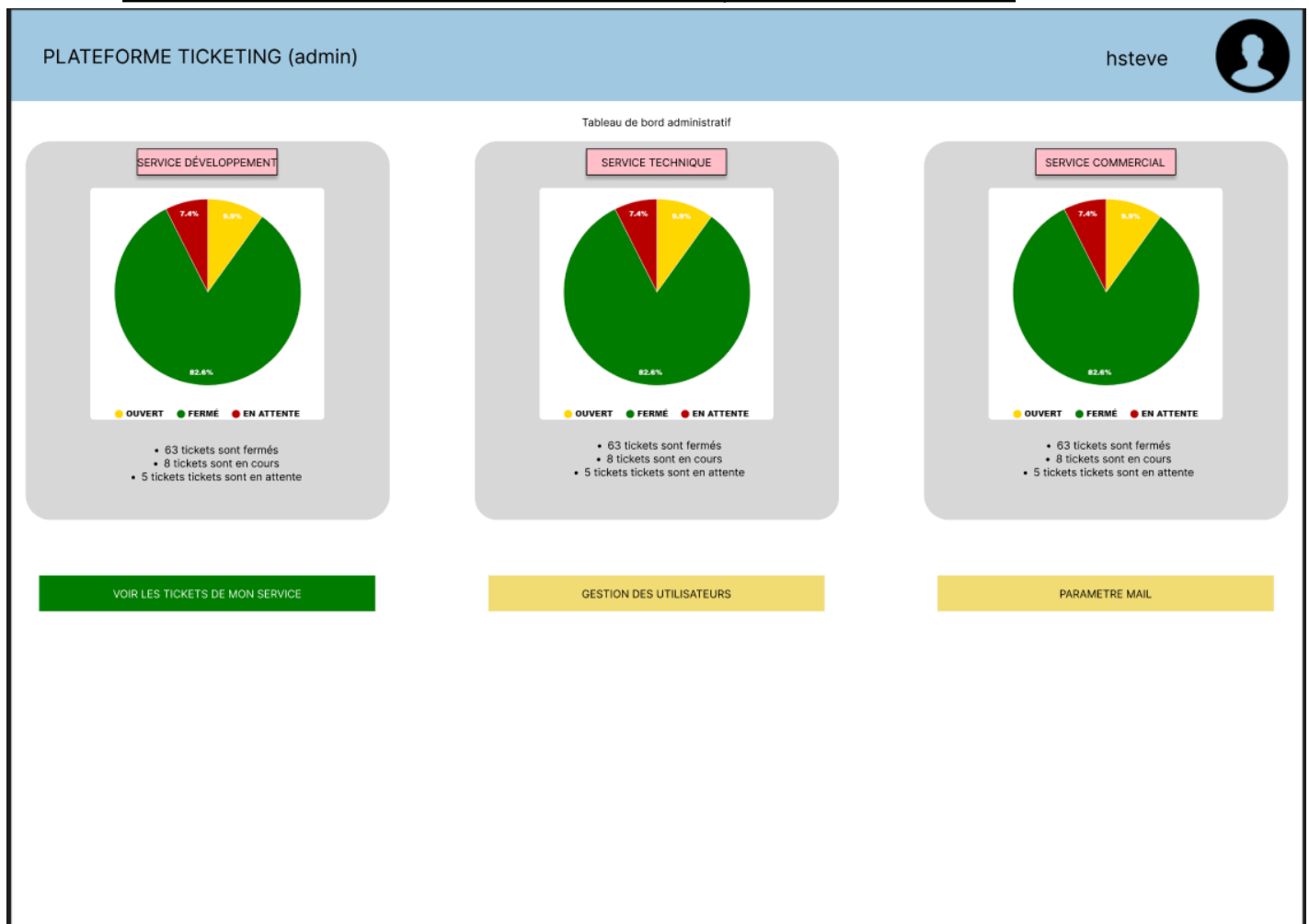
## Annexe 6: Déclencheur 'treatment\_after\_insert':

```
DELIMITER //
```

```
CREATE DEFINER=`ticketing`@`%` TRIGGER `treatment_after_insert`  
AFTER INSERT ON `treatment` FOR EACH ROW BEGIN  
UPDATE ticket  
SET  
    ticket.result_date = (  
        SELECT MAX(treatment.end_date)  
        FROM treatment  
        WHERE treatment.ticket_id = NEW.ticket_id  
        AND treatment.end_date IS NOT NULL  
    ),  
    ticket.result = (  
        CASE  
            WHEN (  
                SELECT COUNT(*)  
                FROM treatment  
                WHERE treatment.ticket_id = NEW.ticket_id  
                AND treatment.end_date IS NOT NULL  
            ) > 0  
            THEN (  
                SELECT treatment.observations  
                FROM treatment  
                WHERE treatment.end_date = (  
                    SELECT MAX(treatment.end_date)  
                    FROM treatment  
                    WHERE treatment.ticket_id = NEW.ticket_id  
                    AND treatment.end_date IS NOT NULL  
                )  
                AND treatment.ticket_id = NEW.ticket_id  
                LIMIT 1  
            )  
            ELSE NULL  
        END  
    )  
END  
)
```

```
WHERE ticket.id = NEW.ticket_id;  
END //
```

## Annexe 7: Interface administrateur, tableau de bord



Cette interface représente la page 9 de l'enchaînement des maquettes (C.f Annexe 1)

## Annexe 8: Requête AJAX se trouvant dans la vue “index.html.twig”:

```
$.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 response dans le Chart
      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 a jour du badge qui affiche le nombre de tickets
      $('#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 (error) { //il peut avoir une erreur
    console.error('Erreur lors de la requête. ' + error.toString());
  }
});
```



## Annexe 9: Procédure “count\_tickets\_service”:

```
DELIMITER //
```

```
CREATE DEFINER='ticketing'@'%' PROCEDURE `count_tickets_service`(  
    IN `serviceId` INT  
)  
LANGUAGE SQL  
NOT DETERMINISTIC  
CONTAINS SQL  
SQL SECURITY DEFINER  
COMMENT ''  
BEGIN  
SELECT  
    COUNT(CASE WHEN t.id NOT IN (SELECT DISTINCT ticket_id FROM treatment)  
        OR latest_treatment.status = 'EN ATTENTE' THEN 1 END) AS in_waiting,  
    COUNT(CASE WHEN latest_treatment.status = 'EN COURS' THEN 1 END) AS in_progress,  
    COUNT(CASE WHEN latest_treatment.status = 'Fermé' THEN 1 END) AS closed  
FROM ticket t  
    LEFT JOIN (  
        SELECT ticket_id, status  
        FROM treatment t1  
        WHERE t1.end_date = (  
            SELECT MAX(t2.end_date)  
            FROM treatment t2  
            WHERE t2.ticket_id = t1.ticket_id  
        )  
    ) latest_treatment ON t.id = latest_treatment.ticket_id  
WHERE t.service_id = serviceId;  
END //
```

## Annexe 10: Workflow GitHub actions:

```
name: Symfony Tests

on: [ push ]

jobs:
  test:
    runs-on: ubuntu-latest

    services:
      mysql:
        image: mysql:8.0
        env:
          MYSQL_ROOT_PASSWORD: rootpassword
          MYSQL_DATABASE: ticketing_test
          MYSQL_USER: ticketing
          MYSQL_PASSWORD: ticketing
        ports:
          - 3306:3306

    steps:
      - name: Checkout code
        uses: actions/checkout@v3

      - name: Set up PHP
        uses: shivammathur/setup-php@v2
        with:
          php-version: '8.1'
          extensions: mbstring, pdo, pdo_mysql, gd, intl
          ini-values: |
            date.timezone=UTC

      - name: Set up Node.js
        uses: actions/setup-node@v3
        with:
          node-version: '20'

      - name: Install Node.js dependencies
        working-directory: web
        run: |
          npm install
          npm run build

      - name: Install Composer
        working-directory: web
        run: |
          curl -sS https://getcomposer.org/installer | php
          mv composer.phar /usr/local/bin/composer
          composer --version

      - name: Install PHP dependencies
        working-directory: web
        run: |
          composer install --prefer-dist --no-progress --no-suggest

      - name: Set up database
        working-directory: web
        run: |
```

```
mysql -h 127.0.0.1 -u root -prootpassword -e "GRANT ALL PRIVILEGES ON
ticketing_test.* TO 'ticketing'@'%';"
mysql -h 127.0.0.1 -u root -prootpassword -e "FLUSH PRIVILEGES;"
mysql -h 127.0.0.1 -u root -prootpassword -e "SET GLOBAL
log_bin_trust_function_creators = 1;"

- name: Run migrations
  working-directory: web
  run: |
    php bin/console doctrine:s:u --env=test --complete --force
    php bin/console doctrine:migrations:migrate --env=test --no-interaction

- name: Run tests
  working-directory: web
  run: |
    php bin/console c:c --env=test
    php bin/phpunit --coverage-html coverage --log-junit junit.xml
  continue-on-error: true
```

## Annexe 11: Script de la base de données

```
/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;  
/*!40101 SET NAMES utf8 */;  
/*!50503 SET NAMES utf8mb4 */;  
/*!40103 SET @OLD_TIME_ZONE=@@TIME_ZONE */;  
/*!40103 SET TIME_ZONE='+00:00' */;  
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0  
*/;  
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;  
/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;
```

```
DROP DATABASE IF EXISTS `ticketing`;  
CREATE DATABASE IF NOT EXISTS `ticketing` /*!40100 DEFAULT CHARACTER SET utf8mb4 COLLATE  
utf8mb4_0900_ai_ci */ /*!80016 DEFAULT ENCRYPTION='N' */;  
USE `ticketing`;
```

```
DROP PROCEDURE IF EXISTS `count_tickets_service`;  
DELIMITER //  
CREATE DEFINER=`ticketing`@`%` PROCEDURE `count_tickets_service`(IN serviceId INT)  
BEGIN  
    SELECT  
        COUNT(CASE WHEN t.id NOT IN (SELECT DISTINCT ticket_id FROM treatment)  
            OR latest_treatment.status = 'EN ATTENTE' THEN 1 END) AS in_waiting,  
        COUNT(CASE WHEN latest_treatment.status = 'EN COURS' THEN 1 END) AS in_progress,  
        COUNT(CASE WHEN latest_treatment.status = 'Fermé' THEN 1 END) AS closed  
    FROM ticket t  
    LEFT JOIN (  
        SELECT ticket_id, status  
        FROM treatment t1  
        WHERE t1.end_date = (  
            SELECT MAX(t2.end_date)  
            FROM treatment t2  
            WHERE t2.ticket_id = t1.ticket_id  
        )  
    ) latest_treatment ON t.id = latest_treatment.ticket_id  
    WHERE t.service_id = serviceId;  
END//  
DELIMITER ;
```

```
DROP PROCEDURE IF EXISTS `count_tickets_user`;  
DELIMITER //  
CREATE DEFINER=`ticketing`@`%` PROCEDURE `count_tickets_user`(IN userId INT)  
BEGIN  
    SELECT  
        (SELECT COUNT(*) FROM treatment WHERE caterer_id = userId) AS n_open,  
        (SELECT COUNT(*) FROM ticket WHERE creator_id = userId) AS n_create,  
        (SELECT COUNT(*) FROM treatment t JOIN ticket ti ON t.ticket_id = ti.id WHERE t.caterer_id = userId  
        AND t.end_date = ti.result_date AND t.`status` = "Fermé") AS n_close;  
END//  
DELIMITER ;
```

```
DROP TABLE IF EXISTS `doctrine_migration_versions`;
CREATE TABLE IF NOT EXISTS `doctrine_migration_versions` (
  `version` varchar(191) COLLATE utf8mb3_unicode_ci NOT NULL,
  `executed_at` datetime DEFAULT NULL,
  `execution_time` int DEFAULT NULL,
  PRIMARY KEY (`version`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3 COLLATE=utf8mb3_unicode_ci;
```

```
DROP TABLE IF EXISTS `mail_configuration`;
CREATE TABLE IF NOT EXISTS `mail_configuration` (
  `id` int NOT NULL AUTO_INCREMENT,
  `login` varchar(150) COLLATE utf8mb4_unicode_ci NOT NULL,
  `password` varchar(150) COLLATE utf8mb4_unicode_ci NOT NULL,
  `smtp_address` varchar(255) COLLATE utf8mb4_unicode_ci NOT NULL,
  `smtp_port` varchar(10) COLLATE utf8mb4_unicode_ci NOT NULL,
  `smtp_tls` tinyint(1) NOT NULL,
  `cc_address` varchar(255) COLLATE utf8mb4_unicode_ci NOT NULL,
  `subject` varchar(255) COLLATE utf8mb4_unicode_ci NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

```
DROP TABLE IF EXISTS `messenger_messages`;
CREATE TABLE IF NOT EXISTS `messenger_messages` (
  `id` bigint NOT NULL AUTO_INCREMENT,
  `body` longtext COLLATE utf8mb4_unicode_ci NOT NULL,
  `headers` longtext COLLATE utf8mb4_unicode_ci NOT NULL,
  `queue_name` varchar(190) COLLATE utf8mb4_unicode_ci NOT NULL,
  `created_at` datetime NOT NULL COMMENT '(DC2Type:datetime_immutable)',
  `available_at` datetime NOT NULL COMMENT '(DC2Type:datetime_immutable)',
  `delivered_at` datetime DEFAULT NULL COMMENT '(DC2Type:datetime_immutable)',
  PRIMARY KEY (`id`),
  KEY `IDX_75EA56E0FB7336F0` (`queue_name`),
  KEY `IDX_75EA56E0E3BD61CE` (`available_at`),
  KEY `IDX_75EA56E016BA31DB` (`delivered_at`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

```
DROP TABLE IF EXISTS `relance`;
CREATE TABLE IF NOT EXISTS `relance` (
  `id` int NOT NULL AUTO_INCREMENT,
  `treatment_id` int NOT NULL,
  `user_id` int NOT NULL,
  `reason` longtext COLLATE utf8mb4_unicode_ci NOT NULL,
  `email` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `reopen` tinyint(1) NOT NULL,
  `relance_date` datetime NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `UNIQU_50BBC126471C0366` (`treatment_id`),
  KEY `IDX_50BBC126A76ED395` (`user_id`),
  CONSTRAINT `FK_50BBC126471C0366` FOREIGN KEY (`treatment_id`) REFERENCES `treatment` (`id`),
  CONSTRAINT `FK_50BBC126A76ED395` FOREIGN KEY (`user_id`) REFERENCES `user` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

```
DROP TABLE IF EXISTS `service`;  
CREATE TABLE IF NOT EXISTS `service` (  
  `id` int NOT NULL AUTO_INCREMENT,  
  `name` varchar(255) COLLATE utf8mb4_unicode_ci NOT NULL,  
  PRIMARY KEY (`id`)  
) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

```
DROP TABLE IF EXISTS `ticket`;  
CREATE TABLE IF NOT EXISTS `ticket` (  
  `id` int NOT NULL AUTO_INCREMENT,  
  `service_id` int DEFAULT NULL,  
  `creator_id` int NOT NULL,  
  `create_date` datetime NOT NULL,  
  `problem` longtext COLLATE utf8mb4_unicode_ci NOT NULL,  
  `result` longtext COLLATE utf8mb4_unicode_ci,  
  `result_date` datetime DEFAULT NULL,  
  `transferred` tinyint(1) NOT NULL,  
  PRIMARY KEY (`id`),  
  KEY `IDX_97A0ADA3ED5CA9E6` (`service_id`),  
  KEY `IDX_97A0ADA361220EA6` (`creator_id`),  
  CONSTRAINT `FK_97A0ADA361220EA6` FOREIGN KEY (`creator_id`) REFERENCES `user` (`id`),  
  CONSTRAINT `FK_97A0ADA3ED5CA9E6` FOREIGN KEY (`service_id`) REFERENCES `service` (`id`) ON  
  DELETE CASCADE  
) ENGINE=InnoDB AUTO_INCREMENT=36 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

```
DROP TABLE IF EXISTS `treatment`;  
CREATE TABLE IF NOT EXISTS `treatment` (  
  `id` int NOT NULL AUTO_INCREMENT,  
  `ticket_id` int NOT NULL,  
  `caterer_id` int NOT NULL,  
  `observations` longtext COLLATE utf8mb4_unicode_ci NOT NULL,  
  `status` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,  
  `start_date` datetime NOT NULL,  
  `end_date` datetime DEFAULT NULL,  
  PRIMARY KEY (`id`),  
  KEY `IDX_98013C31700047D2` (`ticket_id`),  
  KEY `IDX_98013C312CD89E08` (`caterer_id`),  
  CONSTRAINT `FK_98013C312CD89E08` FOREIGN KEY (`caterer_id`) REFERENCES `user` (`id`),  
  CONSTRAINT `FK_98013C31700047D2` FOREIGN KEY (`ticket_id`) REFERENCES `ticket` (`id`)  
) ENGINE=InnoDB AUTO_INCREMENT=11 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

```
DROP TABLE IF EXISTS `user`;
CREATE TABLE IF NOT EXISTS `user` (
  `id` int NOT NULL AUTO_INCREMENT,
  `service_id` int DEFAULT NULL,
  `email` varchar(255) COLLATE utf8mb4_unicode_ci NOT NULL,
  `username` varchar(25) COLLATE utf8mb4_unicode_ci NOT NULL,
  `roles` json NOT NULL,
  `password` varchar(255) COLLATE utf8mb4_unicode_ci NOT NULL,
  `name` varchar(100) COLLATE utf8mb4_unicode_ci NOT NULL,
  `firstname` varchar(25) COLLATE utf8mb4_unicode_ci NOT NULL,
  `active` tinyint(1) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `UNIQ_8D93D649E7927C74` (`email`),
  UNIQUE KEY `UNIQ_8D93D649F85E0677` (`username`),
  KEY `IDX_8D93D649ED5CA9E6` (`service_id`),
  CONSTRAINT `FK_8D93D649ED5CA9E6` FOREIGN KEY (`service_id`) REFERENCES `service` (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

```
DROP TRIGGER IF EXISTS `treatment_after_insert`;
SET @OLDTMP_SQL_MODE=@@SQL_MODE,
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ER
ROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';
DELIMITER //
CREATE DEFINER='ticketing'@'%' TRIGGER `treatment_after_insert` AFTER INSERT ON `treatment` FOR
EACH ROW BEGIN
  UPDATE ticket
  SET
    ticket.result_date = (
      SELECT MAX(treatment.end_date)
      FROM treatment
      WHERE treatment.ticket_id = NEW.ticket_id
      AND treatment.end_date IS NOT NULL
    ),
    ticket.result = (
      CASE
        WHEN (
          SELECT COUNT(*)
          FROM treatment
          WHERE treatment.ticket_id = NEW.ticket_id
          AND treatment.end_date IS NOT NULL
        ) > 0
        THEN (
          SELECT treatment.observations
          FROM treatment
          WHERE treatment.end_date = (
            SELECT MAX(treatment.end_date)
            FROM treatment
            WHERE treatment.ticket_id = NEW.ticket_id
            AND treatment.end_date IS NOT NULL
          )
          AND treatment.ticket_id = NEW.ticket_id
          LIMIT 1
        )
        ELSE NULL
      END
    )
  WHERE ticket.id = NEW.ticket_id;
END//
```

```
DELIMITER ;
SET SQL_MODE=@OLDTMP_SQL_MODE;

DROP TRIGGER IF EXISTS `treatment_after_update`;
SET @OLDTMP_SQL_MODE=@@SQL_MODE,
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ER
ROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';
DELIMITER //
CREATE DEFINER='ticketing'@'%' TRIGGER `treatment_after_update` AFTER UPDATE ON `treatment` FOR
EACH ROW BEGIN
    UPDATE ticket
    SET
        ticket.result_date = (
            SELECT MAX(treatment.end_date)
            FROM treatment
            WHERE treatment.ticket_id = NEW.ticket_id
            AND treatment.end_date IS NOT NULL
        ),
        ticket.result = (
            CASE
                WHEN (
                    SELECT COUNT(*)
                    FROM treatment
                    WHERE treatment.ticket_id = NEW.ticket_id
                    AND treatment.end_date IS NOT NULL
                ) > 0
                THEN (
                    SELECT treatment.observations
                    FROM treatment
                    WHERE treatment.end_date = (
                        SELECT MAX(treatment.end_date)
                        FROM treatment
                        WHERE treatment.ticket_id = NEW.ticket_id
                        AND treatment.end_date IS NOT NULL
                    )
                    AND treatment.ticket_id = NEW.ticket_id
                    LIMIT 1
                )
                ELSE NULL
            END
        )
    WHERE ticket.id = NEW.ticket_id;
END//
DELIMITER ;
SET SQL_MODE=@OLDTMP_SQL_MODE;

/*!40103 SET TIME_ZONE=IFNULL(@OLD_TIME_ZONE, 'system') */;
/*!40101 SET SQL_MODE=IFNULL(@OLD_SQL_MODE, "") */;
/*!40014 SET FOREIGN_KEY_CHECKS=IFNULL(@OLD_FOREIGN_KEY_CHECKS, 1) */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40111 SET SQL_NOTES=IFNULL(@OLD_SQL_NOTES, 1) */;
```