Studi ECF 2023 – 2024

DOCUMENTATION TECHNIQUE

GARAGE V.PARROT

"Réparations fiables, voitures d'occasion exceptionnelles."

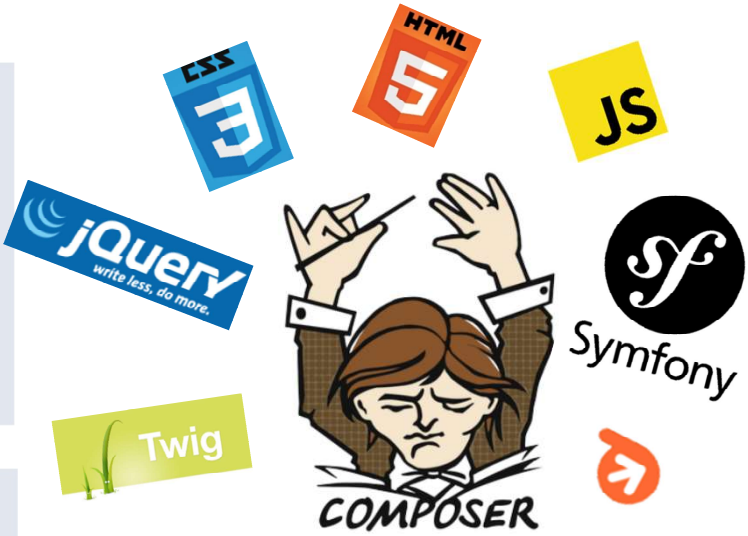
DOCUMENTATION TECHNIQUE

Table des matières

Spécifications techniques

FRONT

- HTML 5
- CSS 3
- TWIG
- Javascript
- JQuery (3.7.0)



SERVEUR LOCAL

XAMPP (8.2.4)

- Maria DB (10.4.28)
- Apache/2.4.56
- PHP (8.2.4)

BACK

- Composer (2.5.8)
- Symfony (6.3)
- Bundles :
  Doctrine
  VichUploader
  Knp-paginator (6.2)
  Form
  Make-bundle (dev)
  Security bundle
  Password-hasher
  Validator
  Annotations
  Rate-limiter
  Profiler (dev)
  Extension intl (heroku)

SERVEUR PRODUCTION

HEROKU

- MariaDB
- Apache
- PHP

ENVIRONNEMENT DE TRAVAIL

- Windows 10
- Visual Studio Code
- Figma (Wireframes & Mockup)
- Draw.io Integration version 1.6.6 (UML)
- Github
- Trello

Diagramme de cas d'utilisation - Garage V. PARROT

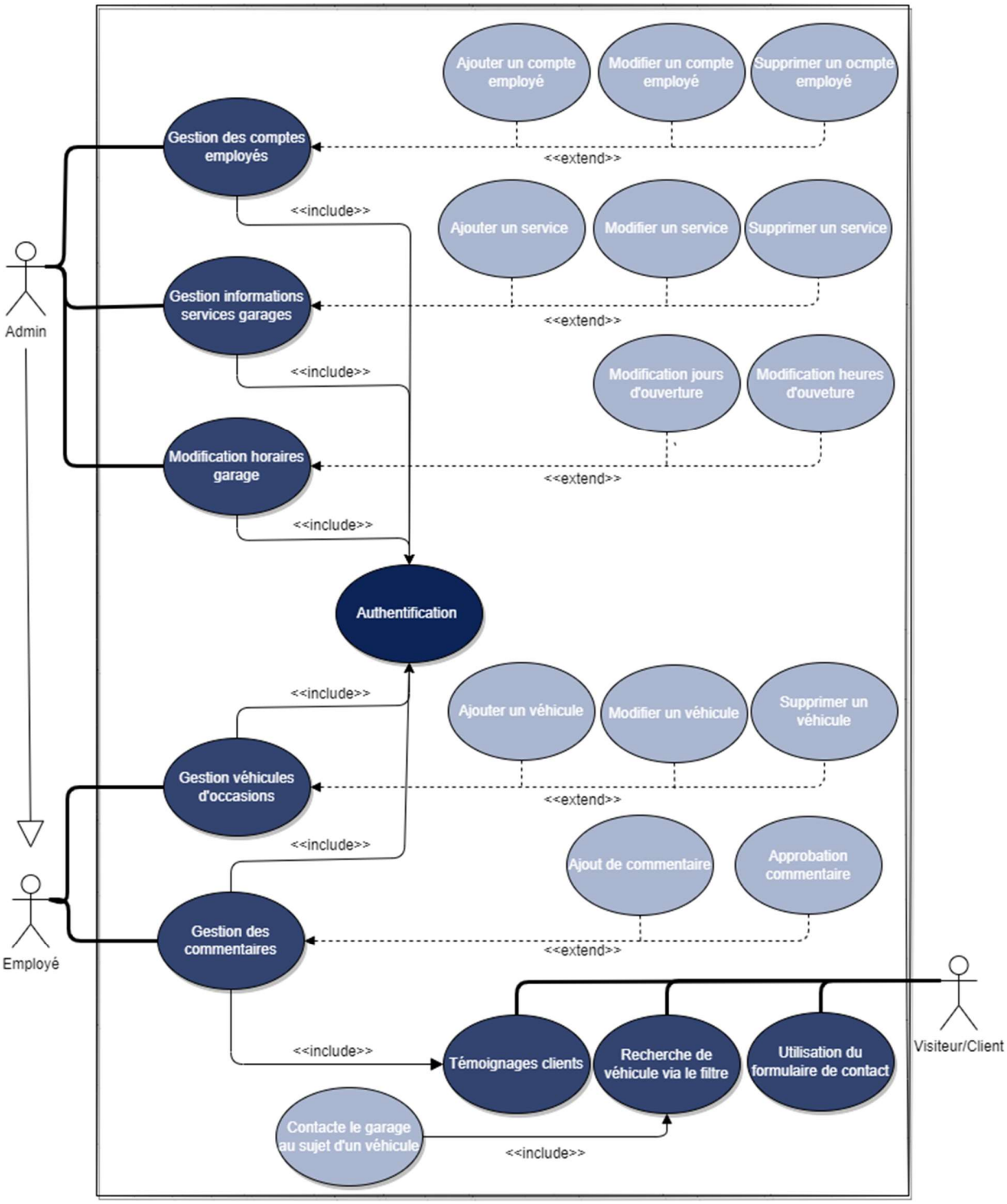
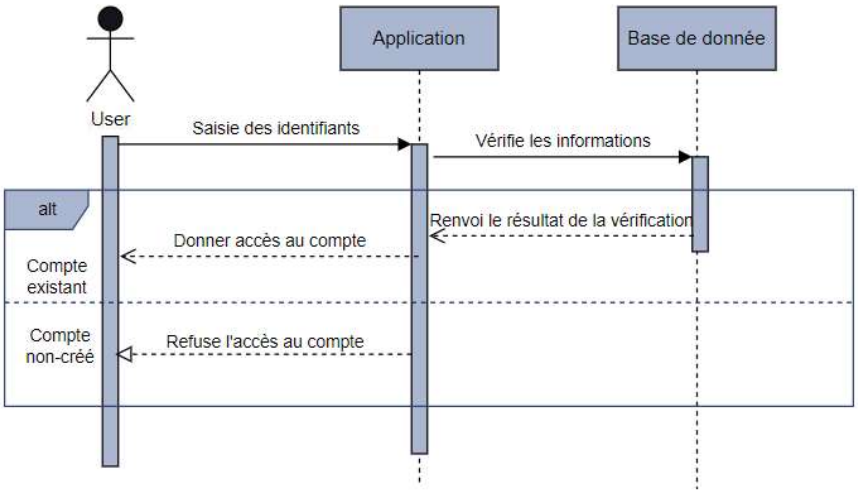
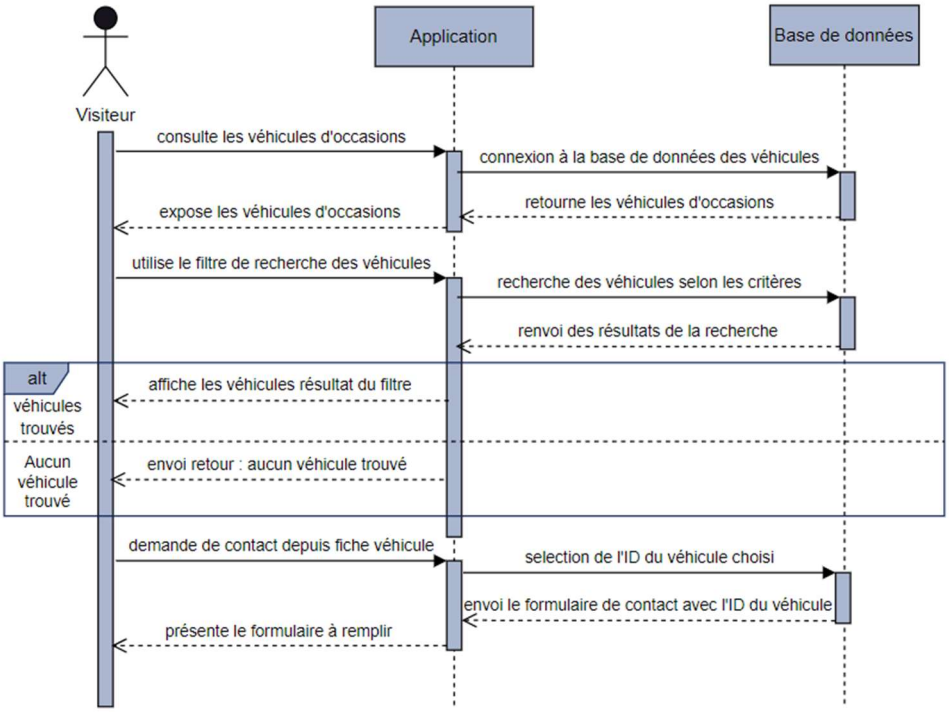


Diagramme de séquence - Garage V. PARROT

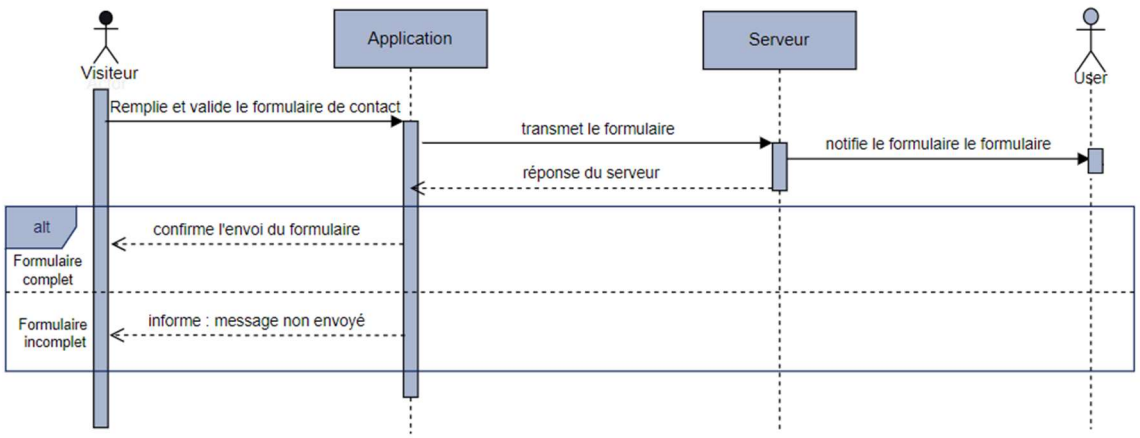
US1. Se connecter



US4. Exposer les voitures d'occasion, US5. Filtrer la liste des véhicules



US6. Permettre de contacte l'atelier



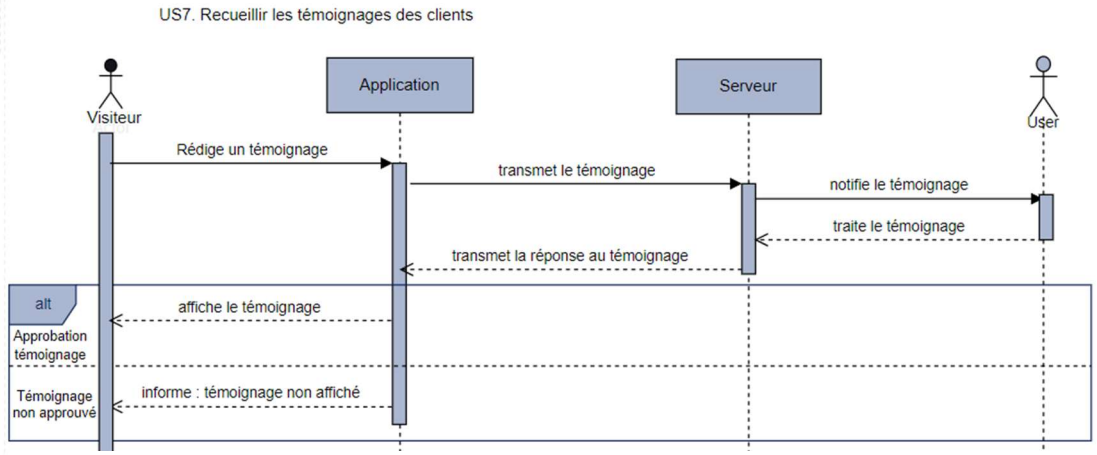
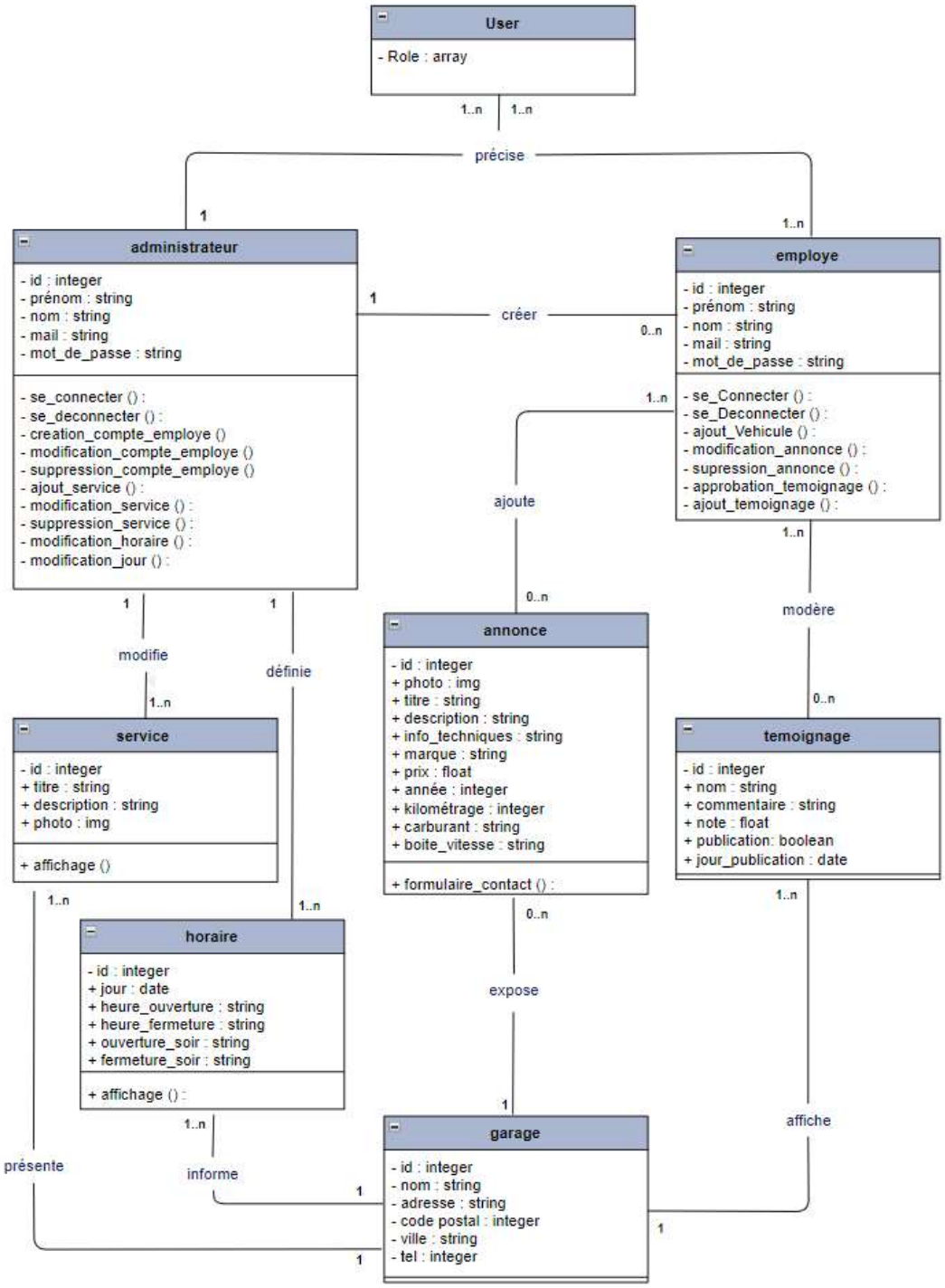


Diagramme de classe - Garage V. PARROT



Mesure de sécurité

Le framework Symfony de PHP propose le « bundle Security » qui intègre un ensemble de fonctionnalité permettant la gestion de la sécurité de l'application. Nous les exploiterons en suivant également les recommandations de la CNIL (Commission Nationale de l'Informatique et des Libertés) ainsi que les bonnes pratiques en matière de sécurité.

L'authentification

- Renforcement de la politique de sécurité lors de la création de mot de passe par l'utilisateur. Conformément à l'une des recommandations de la CNIL en matière de protection des mots de passe, nous allons soumettre nos utilisateurs à l'utilisation d'un mot de passe d'un minimum de 12 caractères comprenant des majuscules, des minuscules, des chiffres et des caractères spéciaux. Dans notre cas, ils ne seront visibles que dans la documentation.

- De plus, avec un *Entity Listener*, les mots de passes de tous les utilisateurs de l'application sont systématiquement et automatiquement hashés, à la fois dans l'application et dans la base de données. Notre administrateur fournira donc le mot de passe à ses utilisateurs par un biais en dehors de notre application afin de les maintenir secrets.

Protection contre les attaques de forces brutes

- Cette attaque consiste à tenter de se connecter avec acharnement à l'application en cherchant à devenir le mot de passe. L'utilisateur utilisera successivement des mots de passe
- La première manière de s'en protéger et, comme mentionné ci-dessus, d'imposer des mots de passe long et complexe.
- La deuxième manière consiste tout simplement à limiter dans le temps le nombre de tentative de connexion grâce au composant Rate Limiter. Ainsi on peut configurer le nombre de tentative : au bout de trois essais sans succès, l'utilisateur ne pourra plus tenter de se connecter pendant 30 minutes. C'est une bonne manière de démotiver les utilisateurs malveillants à retenter.
  Cependant, dans le cadre de notre application, nous n'irons pas plus loin dans la « sanction » car il est tout à fait possible qu'un employé du garage ou M. Parrot puisse malheureusement commettre des erreurs de saisies. Ainsi, ils pourront retenter leur connexion ultérieurement.

Les autorisations

- La gestion des droits d'utilisateur sont directement assurés par un seul administrateur qui définis le rôle dudit utilisateur à chaque création de compte.

- Uniquement l'administrateur possède l'accès à toutes les pages de l'application. Les autres utilisateurs sont soumis à une restriction de certaines pages et certaines fonctionnalités de l'application grâce aux fonctions de permissions proposées par EasyAdmin.

Protection contre les injections SQL

- Tout d'abord le Bundle Validator rend obligatoire la saisie de données par un utilisateurs grâce au système de « contraintes ». En plus, il permet aussi de soumettre les saisies à certaines contraintes pour s'assurer que ces dernières soient correctes.
- Ensuite, l'ORM Doctrine, utilisé pour interagir avec la base de données propose des mesures de sécurité pour protéger l'application des injections SQL : il utilise pour cela une identification précise des paramètres préparés dans les déclarations SQL.

Protection contre le Cross Site Scripting (XSS)

- L'utilisation du moteur de template TWIG permet de se prémunir contre le Cross Site Scipting (XSS). Cette manœuvre consiste à injecter du code javascript malveillant dans une page web par le biais d'une saisie d'utilisateur. Les conséquentes peuvent être grave sur l'application mais aussi entraîner des répercussions sur d'autres utilisateurs.
- C'est notamment grâce à sa syntaxe entre double accolades, qui permet l'échappement des données que TWIG sert de bouclier contre les XSS. Autrement dit, les caractères spéciaux sont convertis en entités HTML dans le navigateur et non pas en code javascript.

Les formulaires

- Le CSRF (Cross Site Request Forgery) permet à un utilisateur malveillant de faire en sorte qu'un utilisateur de l'application soumettent des données à son insu en utilisant le code source du formulaire de l'application. Même si, par exemple notre formulaire « Témoignage » est soumis à une vérification avant publication de M. PARROT et de son équipe, il est important de se prémunir contre ce genre de manœuvre.

  Pour s'en prémunir, on va ajouter dans nos formulaires des valeurs secrètes qui prendront la forme « type = 'hidden' ».

  Dans son volet Security, Symfony ajoute automatiquement une protection contre le CSRF avec l'implémentation d'un Token lors de la validation d'un formulaire.