

NOM	Dubois
Prénom	Adèle
Date de naissance	17/09/1991

Copie à rendre

Graduate Développeur

(Android, Angular, Flutter, Front End, Full Stack, IOS, PHP/Symfony)

Documents à compléter et à rendre

Activité – Type 1 : Développer la partie front-end d'une application web ou web mobile en intégrant les recommandations de sécurité

Lien dépôt Github : <https://github.com/adele-dbs/garagevparrot>

Lien version en ligne : <http://garave.v.parrot.go.yj.fr/>

Lien Trello : <https://trello.com/b/f9oSziRT/ecf>

Définition des besoins et planification :

Pour commencer, j'ai analysé les besoins.

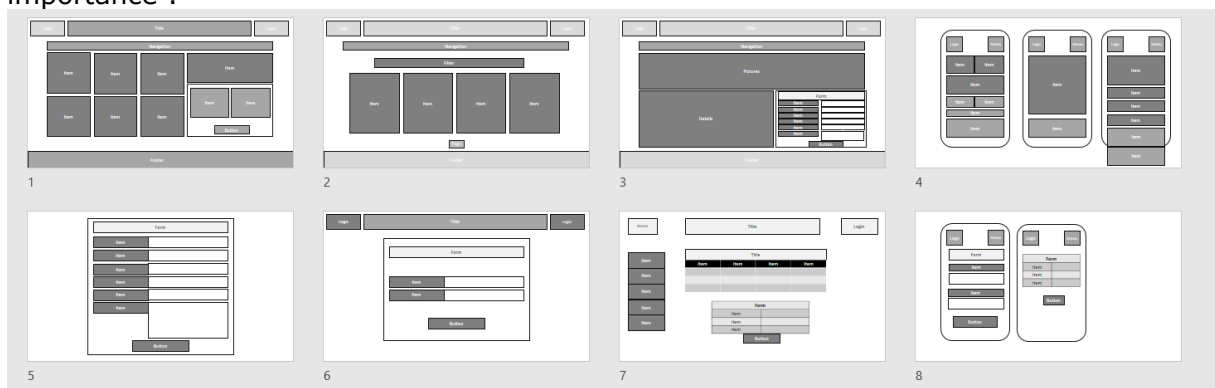
J'ai ensuite planifié mon projet en utilisant l'application Trello. Elle m'a permis de suivre l'avancement grâce au tableau Kanban.

Maquettage :

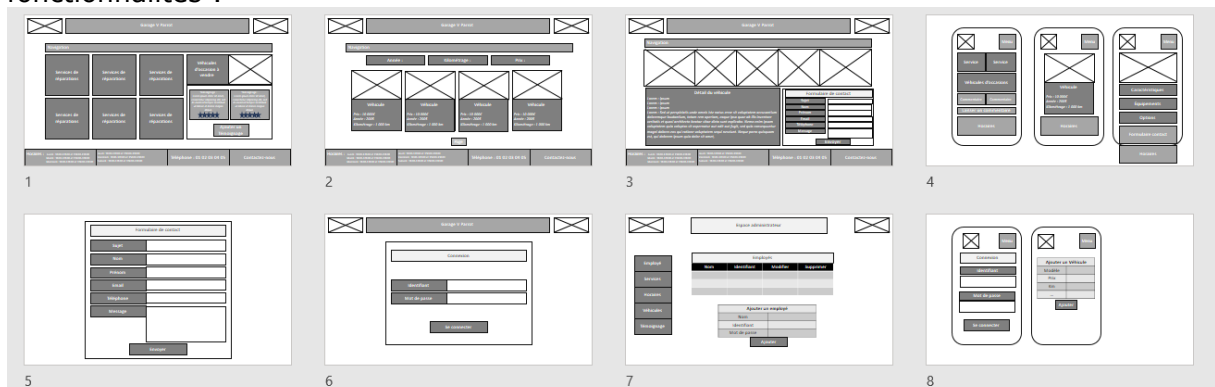
J'ai ensuite maqueté mon application.

J'ai commencé par définir les pages, puis j'ai ajouté les blocs pour commencer le zoning.

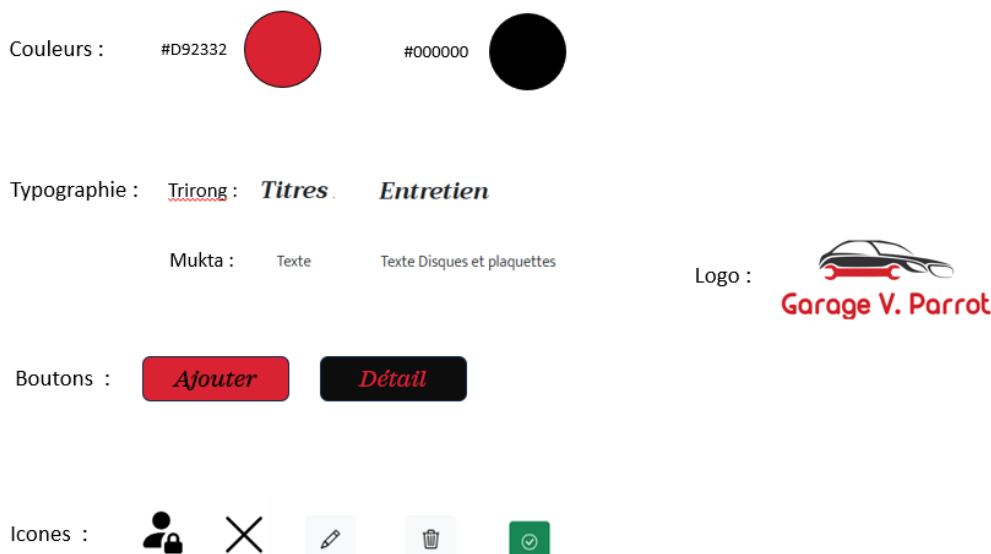
J'ai ensuite nommé ces blocs et j'ai ajouté des nuances de gris en fonction de leur importance :



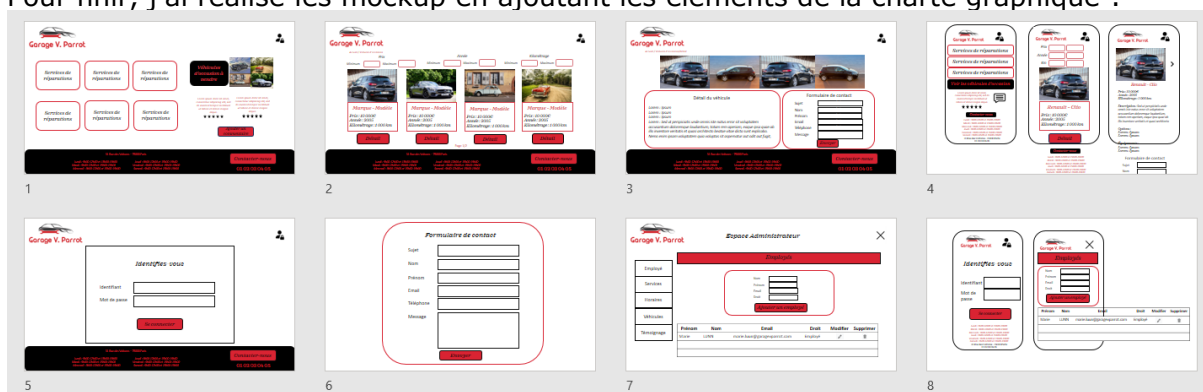
J'ai ensuite réalisé les wireframes en ajoutant les principaux éléments ainsi que les fonctionnalités :



Pour ajouter des couleurs ainsi que les polices d'écritures, j'ai défini la charte graphique :



Pour finir, j'ai réalisé les mockup en ajoutant les éléments de la charte graphique :



J'ai utilisé le fil d'Ariane pour ajouter de l'ergonomie.

Concernant le visuel, j'ai essayé de prendre en compte les tendances actuelles pour proposer un style épuré.

Réalisation de l'interface :

HTML :

J'ai réalisé deux layouts : un pour la partie client et un autre pour la partie utilisateurs afin d'obtenir une trame, de limiter les redondances dans le code et de faciliter la maintenance.

J'ai ensuite créé les templates des pages : un fichier par page.

Style :

J'ai utilisé CSS ainsi que le Framework Bootstrap pour faciliter la mise en page et l'ajout de style.

Référencement :

Pour améliorer le référencement, j'ai structuré mon site en utilisant un balisage hiérarchique.

J'ai également utilisé des balises <meta> :

```
14 <title><?= $titre ?></title>
15 <meta <?= $meta ?>>
```

```
2 $titre = 'Garage V.Parrot - Véhicules d\'Occasion';
3 $meta = 'name="description"
4 content="Lorem ipsum dolor sit amet, consectetur adipiscing elit."';
```

Responsive Design :

Pour le rendre responsive, j'ai utilisé :

- la balise <meta> Viewport :

```
11 <meta
12     name="viewport"
13     content="width=device-width, initial-scale=1">
```

- les unités de longueurs relatives, celles basées sur le viewport et les pourcentages
- les media queries :

```
156 @media screen and (max-width: 576px) {
157     #login-f {
158         width : 100%;
159     }
```

- le framework Bootstrap

```
9 <section class="col-12 col-sm-3">
```

Git :

J'ai réalisé des commits réguliers grâce à Git et Github afin d'effectuer des sauvegardes à chaque étape de l'avancement du projet.

Sécurité :

J'ai sécurisé les formulaires en précisant le type de données (number, ...) et les caractères acceptés grâce au pattern, et en limitant le nombre de caractères avec les attribut de input : minlength et maxlength :

```
pattern="[a-z0-9._%+\-]+\@[a-z0-9.\-]+\.[a-z]{2,}$" required>
```

```
pattern="[a-zA-Z0-9]+" maxlength="20" required>
```

```
<input type="number" class="form-control" name="addrating" id="addrating" min=0 max=5 required>
```

Et en utilisant des listes déroulantes afin d'éviter les saisies libres :

```
74 <label for="addstaffright">Droit : </label>
75 <select class="form-select" aria-label="Default select example" name="addstaffright" id="addstaffright" required>
76     <?php foreach ($rights as $right): ?>
77         <option value="<?= $right->getRightId() ?>"><?= $right->getRightId() ?> - <?= $right->getRightName() ?></option>
78     <?php endforeach; ?>
79 </select>
```

JS : dynamique :

Pour rendre l'application dynamique, j'ai utilisé Javascript.
Pour afficher des messages d'alerte :

```
views > JS login-messages.js > ...
1 let form = document.querySelector('form')
2
3 form.addEventListener('submit', (event) => {
4   for(var count=0; count<form.elements.length; count++) {
5     switch (form.elements[count].name) {
6       case 'email':
7         if (form.elements[count].value === '') {
8           alert('L\'email est obligatoire')
9           event.preventDefault();
10        }
11        break;
12       case 'password':
13         if (form.elements[count].value === '') {
14           alert('Le mot de passe est obligatoire')
15           event.preventDefault();
16        }
17        break;
18      }
19    }
20  })
```

Pour masquer/afficher des formulaires :

```
views > JS display-form.js > ...
1 let staffform = document.getElementById('staffform');
2 let btnDisplayAddForm = document.getElementById('buttonaddstaffform');
3 let btnHideAddForm = document.getElementById('buttonAddStaff');
4
5 function displayForm() {
6   staffform.style.display = "block";
7 };
8
9 function hideForm() {
10  staffform.style.display = "none";
11 };
12
13 btnDisplayAddForm.addEventListener('click', displayForm);
14 btnHideAddForm.addEventListener('submit', hideForm);
```

Réalisation de l'interface utilisateur avec une solution de gestion de contenu :

J'ai ensuite créé un espace utilisateur.

J'ai commencé par créer une page de connexion avec un formulaire.

J'ai ensuite créé une page administrateur et une page employé.

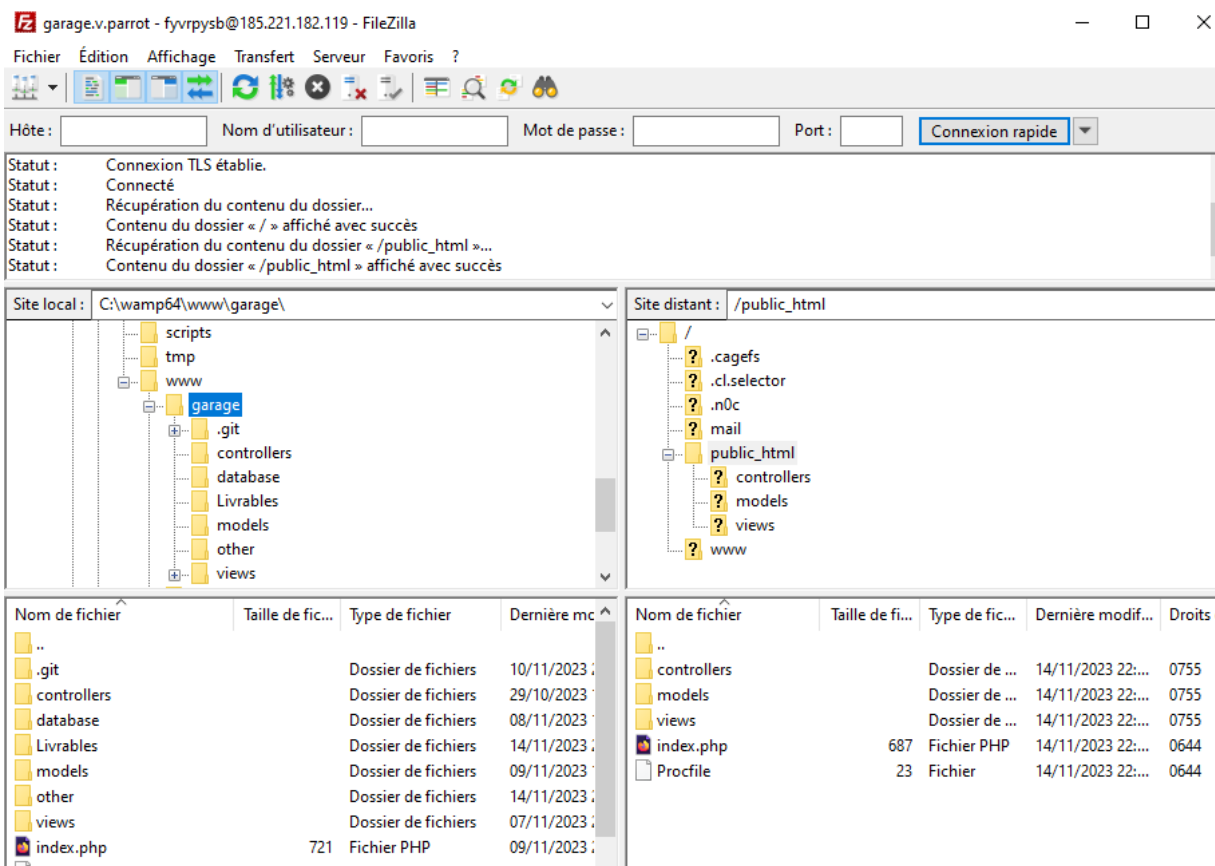
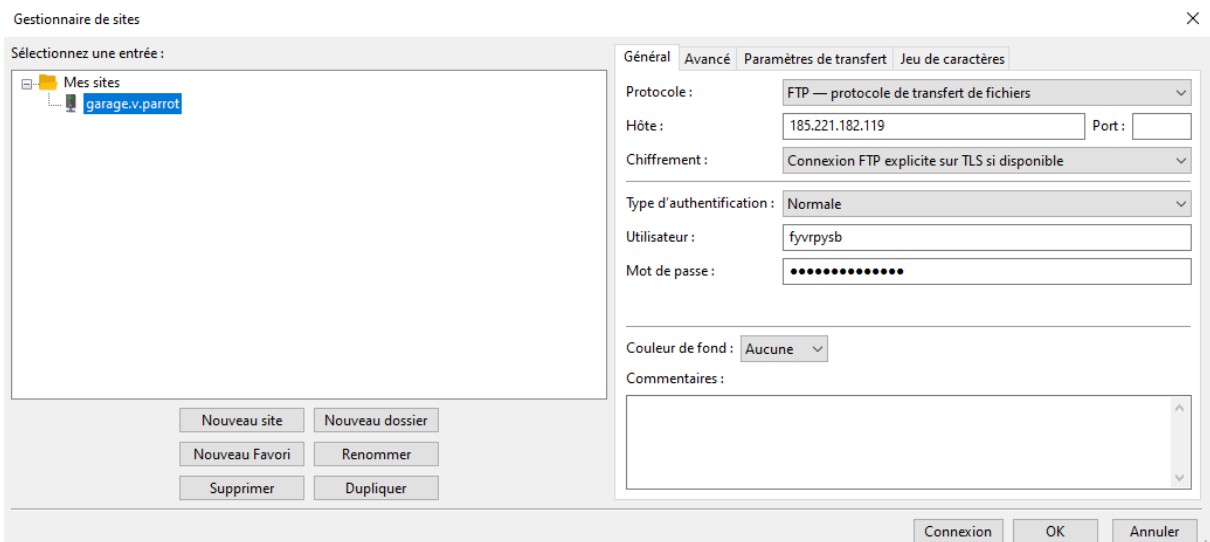
Pour gérer les différents contenus, j'ai créé des sous-parties avec un tableau pour afficher les informations et un formulaire pour les modifier ou en ajouter d'autres.

J'ai également ajouté un bouton pour supprimer une ligne.

Déploiement

J'ai déployé mon projet avec World Lite de Planet Hoster.

J'ai utilisé le client FTP FileZilla pour importer les fichiers :



Pour la base de données, j'ai créé la base de données et l'utilisateur sur NOC :

<https://mg.n0c.com/login>

Utilisateur : fyvrpysb




Mot de passe : B3vdcwGp4VHaCY

Bases de données i

Gérez vos bases de données SQL. Créez une nouvelle base de données, ajoutez un utilisateur à une base de données, modifiez les privilèges des utilisateurs et bien plus.

BASES DE DONNÉES ACTUELLES

[Créer](#)



BASES DE DONNÉES	TAILLE ↓	UTILISATEURS AVEC PRIVILÈGES	ACTIONS
fyvrpysb_garage	39 kB	fyvrpysb_user_garage  	

Affichage 1 - 1 sur 1 éléments

|< < > >|

UTILISATEURS ACTUELS

[Créer](#)

UTILISATEURS	ACTIONS
fyvrpysb_user_garage	 

Affichage 1 - 1 sur 1 éléments

|< < > >|

Puis j'ai importé les tables via PHPAdmin.

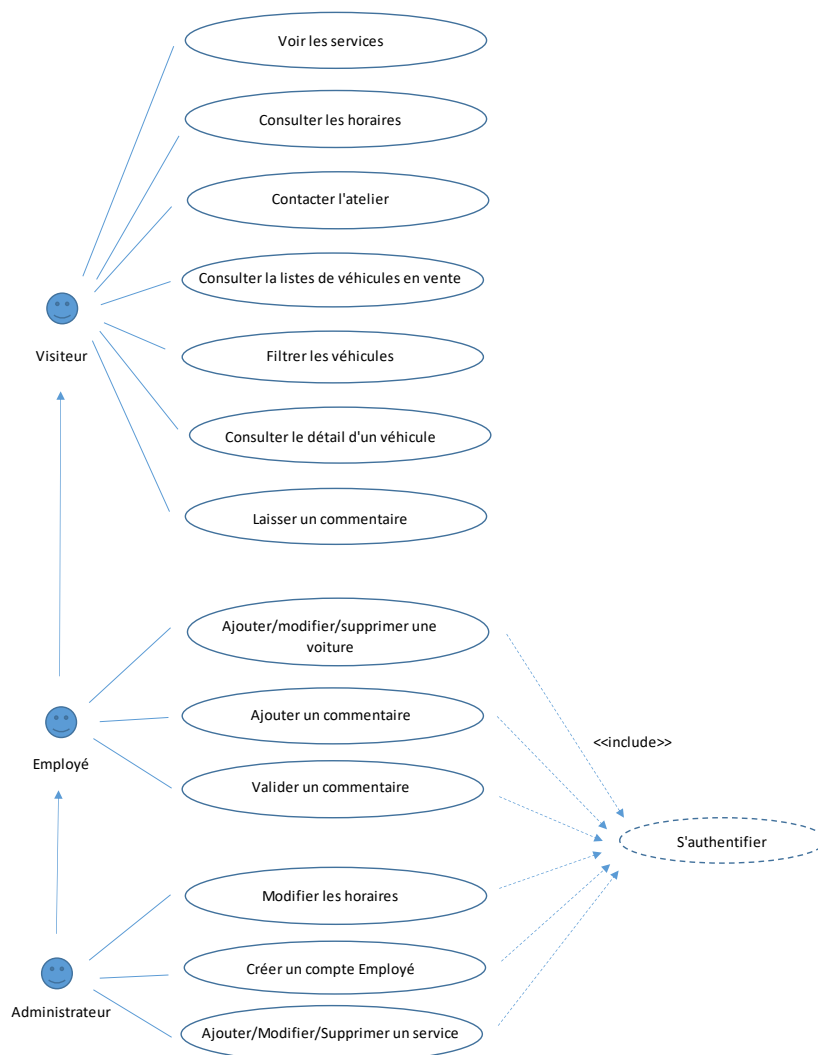
Activité – Type 2 : Développer la partie back-end d'une application web ou web mobile en intégrant les recommandations de sécurité

Création de la base de données :

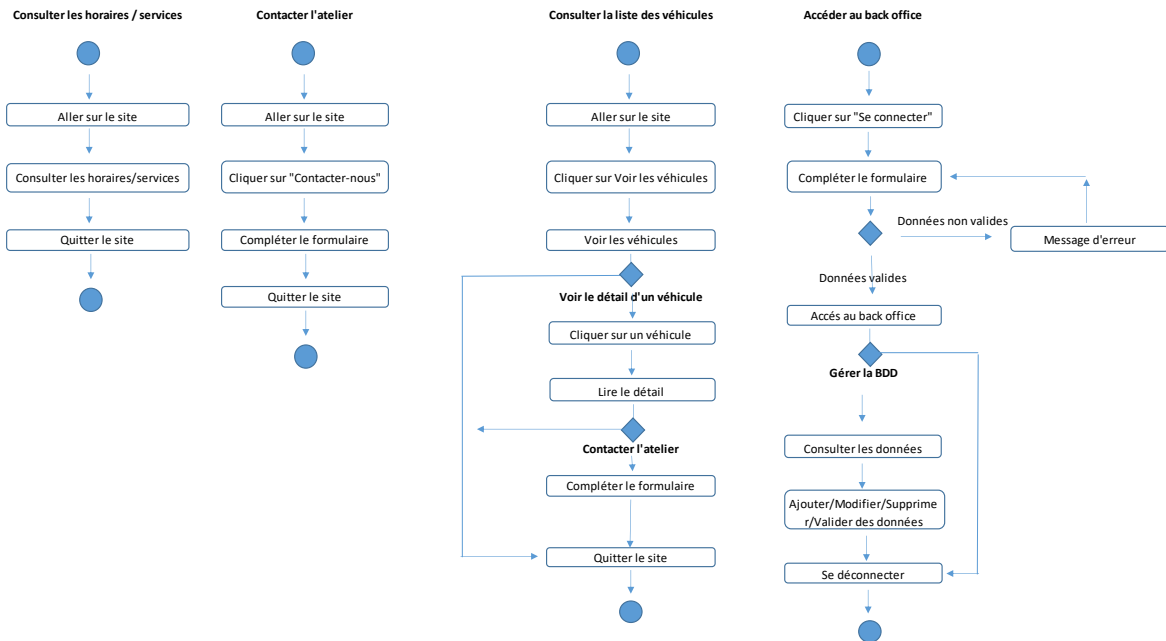
Afin de déterminer toutes les fonctionnalités de l'application, j'ai réalisé une analyse du cahier de charges.

Pour cela, j'ai utilisé 3 diagrammes de comportement :

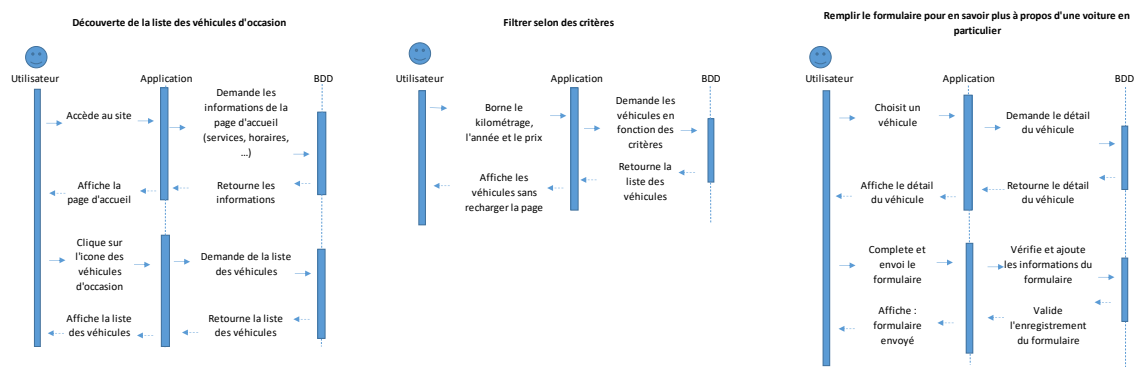
- Le diagramme de cas d'utilisations afin de lister les fonctionnalités de l'application :



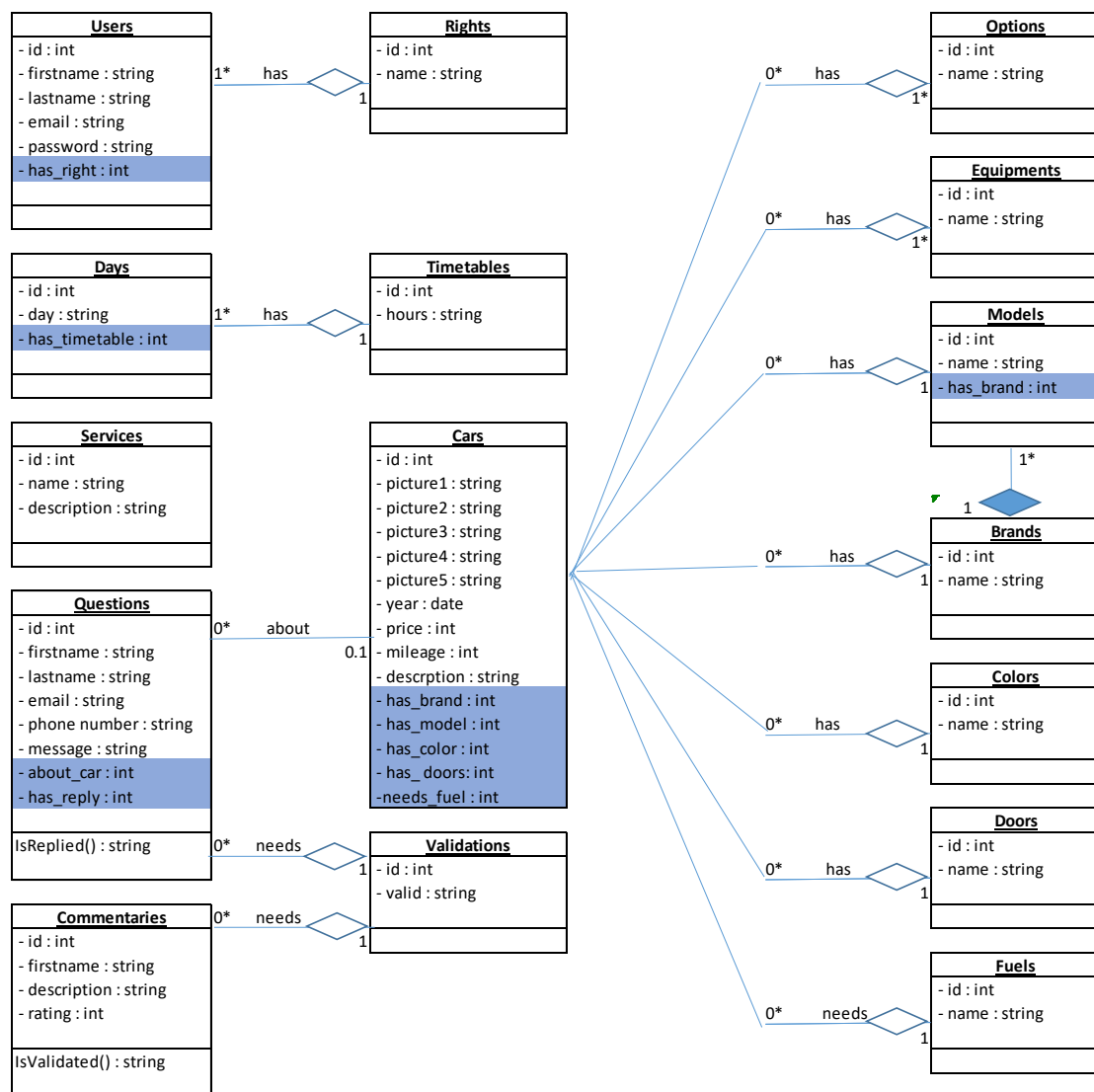
- Le diagramme d'activité afin de découper les fonctionnalités en les étapes :



- Le diagramme de séquence pour illustrer les interactions entre les différents acteurs :



J'ai ensuite réalisé un diagramme de classes qui m'a permis de représenter les différentes classes, leurs attributs et leurs méthodes ainsi que les relations entre elles, afin d'obtenir un modèle conceptuel de données.



Pour finir, j'ai créé la base de données.

J'ai utilisé le serveur local WAMP qui m'a permis d'avoir accès à MySQL.

Dans un premier temps, je me suis connectée à MySQL à l'aide du terminal de commande de Windows (CMD) :

```

C:\Users\Adele>cd C:\wamp64\bin\mysql\mysql8.0.31\bin
C:\wamp64\bin\mysql\mysql8.0.31\bin>mysql -u root -p
  
```

J'ai ensuite créé la base de données :

```

mysql> CREATE DATABASE garage;
Query OK, 1 row affected (0.18 sec)
  
```

J'ai créé un utilisateur pour la base de donnée :

```

mysql> CREATE USER 'user_garage'@'localhost' IDENTIFIED BY 'D1felkf9Za';
Query OK, 0 rows affected (6.43 sec)
  
```

Je lui ai ensuite donné des droits :

```

mysql> GRANT SELECT, INSERT, UPDATE, DELETE ON garage.* TO 'user_garage'@'localhost';
Query OK, 0 rows affected (0.83 sec)
  
```

Puis j'ai réalisé un script pour créer les tables ainsi que les contraintes :

```
mysql> CREATE TABLE garage.rights (
-> id INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
-> name VARCHAR(50) NOT NULL
-> );
Query OK, 0 rows affected, 1 warning (1.91 sec)
```

J'ai ensuite alimenté la base de données :

```
mysql> INSERT INTO garage.rights (name) VALUES
-> ('ADMIN'),
-> ('STAFF');
Query OK, 2 rows affected (0.15 sec)
Enregistrements: 2 Doublons: 0 Avertissements: 0
```

Les mots de passe ont été chiffrés en utilisant la fonction password_hash et l'algorithme BCRYPT afin de renforcer la sécurité.

Voici les mots de passe :

```
vincent.parrot@garagevparrot.com : p4$$w0rd - Admin
marie.luun@garagevparrot.com : P&ssW&rd4!p- User
```

Développement des composants d'accès aux données :

J'ai ensuite relié la base de données à l'application en local :

```
models > Model.php > {} Model
1  <?php
2
3  trait Model
4  {
5      private $pdo = null;
6
7      public function __construct()
8      {
9          $hostname = 'localhost';
10         $username = 'user_garage';
11         $password = 'D1felkf9Za';
12         $database = 'garage';
13
14         try {
15             //SQL Injection
16             $options = [
17                 PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
18             ];
19             $this->pdo = new PDO("mysql:host=$hostname;dbname=$database", $username, $password, $options);
20         } catch (PDOException $PDOException) {
21             echo 'Impossible de se connecter à la base de donnée';
22             exit('Erreur : '.$PDOException->getMessage());
23         }
24     }
25 }
```

Lors du déploiement, j'ai modifier les informations de connexion:

```
7      public function __construct()
8      {
9          $hostname = 'localhost';
10         $username = 'fyvrpysb_user_garage';
11         $password = 'D1felkf9Za';
12         $database = 'fyvrpysb_garage';
```

J'ai ensuite créé deux classes pour chaque table comme le recommandent les bonnes pratiques : la première correspond à une ligne de la table et la second à l'ensemble de la table.

- 🐘 Question.php
- 🐘 Questions.php
- 🐘 Right.php
- 🐘 Rights.php
- 🐘 Service.php
- 🐘 Services.php
- 🐘 Timetable.php
- 🐘 Timetables.php
- 🐘 User.php
- 🐘 Users.php
- 🐘 Validation.php
- 🐘 Validations.php

```
models > 🐘 Right.php > ...
1  <?php
2
3  require_once('models/Model.php');
4
5  class Right
6  {
7      use Model;
8
9      private int $id;
10     private string $name;
11
12     public function getRightId()
13     {
14         return $this->id;
15     }
16
17     public function getRightName()
18     {
19         return $this->name;
20     }
21 }
```

Puis j'ai créé des fonctions pour accéder aux données via des requêtes SQL :

```
9     public function getRights ()
10    {
11        $stmt = $this->pdo->query('SELECT * FROM rights');
12        $rights = [];
13        while ($right = $stmt->fetchObject('Right')) {
14            $rights[] = $right;
15        }
16        return $rights;
17    }
```

J'a également utilisé des jointures :

```
17     public function getCommentaries ()
18    {
19        $commentaries = $this->pdo->query('SELECT commentaries.id id, commentaries.firstname name,
20        commentaries.commentary commentary, commentaries.rating rating, commentaries.valid_id valid,
21        validations.id validid, validations.valid validname
22        FROM commentaries
23        INNER JOIN validations ON validations.id = commentaries.valid_id');
24        return $commentaries;
25    }
```

Et des conditions :

```
13 public function getDayDetailById (int $id)
14 {
15     $stmt = $this->pdo->prepare('SELECT * FROM days WHERE id = ?');
16     $dayById = null;
17     if ($stmt->execute([$id])) {
18         $dayById = $stmt->fetchObject('Day');
19         if(!is_object($dayById)) {
20             $dayById = null;
21         }
22     }
23     return $dayById;
24 }
```

J'ai ensuite utilisé des boucles pour afficher les données :

```
14 <?php foreach ($services as $service): ?>
15     <div class="col-md-6">
16         <div class="card h-100 services-card">
17             <div class="card-body">
18                 <h5 class="card-title"><?= $service->getServiceName() ?></h5>
19                 <p class="card-text"><?= $service->getServiceDescription() ?></p>
20             </div>
21         </div>
22     </div>
23 <?php endforeach; ?>
```

J'ai structuré mon projet selon le modèle MCV.

En effet, j'ai séparé la logique applicative, l'interface graphique et le traitement des actions de l'utilisateur en créant 3 dossiers : Model, View, Controller.

Dans la partie Views, on retrouve les templates des pages, les layouts, le style en CSS, et les fichiers JavaScript.

La partie Models regroupe les classes et le fichier d'accès à la base de données (Model).

Et dans la partie contrôleur, on trouve le contrôleur secondaire. Le contrôleur principal étant à la racine du projet (index.php).

J'ai alimenté le fichier index.php en lien avec le contrôleur secondaire afin d'effectuer le routing entre les pages :

```
3 require_once 'controllers/controller.php';
4
5 $controller = new Controller();
6
7 if (isset($_GET['page'])) {
8     switch ($_GET['page']) {
9         case 'cars':
10             $controller->listCars();
11             break;
12         case 'cardetail':
13             $controller->detailCars();
14             break;
15         case 'login':
16             $controller->login();
17             break;
18         case 'logout':
19             $controller->logout();
20             break;
21         case 'admin':
22             $controller->userInterfaceAdmin();
23             break;
24         case 'staff':
25             $controller->userInterfaceStaff();
26             break;
27         default:
28             $controller->home();
29             break;
30     }
31 } else {
32     $controller->home();
33 }
```

Développement de la partie back-end :

J'a ensuite créé la partie back-end.

J'ai d'accord réalisé une page de connexion avec un formulaire vérifiant des informations de connexion ainsi que les droits des utilisateurs :

```

16 public function getLogin (string $email, string $password)
17 {
18     function validUserdonnees($donnees) {
19         $donnees = trim($donnees);
20         $donnees = stripslashes($donnees);
21         $donnees = htmlspecialchars($donnees);
22         return $donnees;
23     }
24
25     $email = validUserdonnees($_POST["email"]);
26     $password = validUserdonnees($_POST["password"]);
27
28     if($email != "" && $password != "") {
29         $stmt = $this->pdo->prepare('SELECT * FROM users WHERE email = :email');
30         $stmt->setFetchMode(PDO::FETCH_CLASS, 'User');
31         $stmt->bindValue(':email', $email);
32         if ($stmt->execute()) {
33             $user = $stmt->fetch();
34             if ($user != false && password_verify($password, $user->getPassword())) {
35                 if ($user->getRightId() === 1) {
36                     $_SESSION["autoriser"]="oui";
37                     header('Location: ?page=admin');
38                 } else {
39                     if ($user->getRightId() === 2) {
40                         $_SESSION["autoriser"]="oui";
41                         header('Location: ?page=staff');
42                     } else {
43                         require_once 'views/login.php';
44                         ?>
45                         <script type="text/javascript"> alert('Vous n\' être pas autorisé à vous connecter') </script>

```

Le formulaire de connexion permet d'ouvrir une session afin d'accéder à l'espace utilisateur :

```

public function login()
{
    session_start();
    if(isset($_POST['email']) && isset($_POST['password']))
    {
        $_SESSION['email'] = $_POST['email'];
        $user = $this->userObject->getLogin($_POST['email'], ($_POST['password']));
    }
}

```

J'ai ensuite affiché les tables par l'intermédiaire de tableaux, grâce à l'utilisation de boucles :

```

178 <?php foreach ($services as $service): ?>
179 <tr>
180 <td><?= $service->getServiceName() ?></td>
181 <td><?= $service->getServiceDescription() ?></td>
182 <td>

```

J'ai également créé une fonction de déconnexion :

```

211 public function logout()
212 {
213     session_start();
214     $_SESSION = [];
215     session_destroy();
216     header("location: ?page=login");
217     exit();
218 }

```

Élaboration et mise en œuvre des composants de gestion de contenu :

J'ai ensuite ajouté des formulaires pour ajouter des données :

```

155 <form action="" method="POST" id="addServiceForm">
156 <div class="form-group">
157 <label for="addname">Nom : </label>
158 <input type="text" name="addname" class="form-control" id="addname" required>
159 <label for="adddescription">Description: </label>
160 <input type="text" name="adddescription" class="form-control" id="adddescription" required>
161 </div>
162 <button type="submit" class="btn backendButton" id="buttonAddService">Ajouter</button>
163 </form>

```

Ainsi que les fonctions pour ajouter des données dans la base de données :

```

26 public function addService (string $addname, string $adddescription)
27 {
28     function validAddServicedonnees($donnees) {
29         $donnees = trim($donnees);
30         $donnees = stripslashes($donnees);
31         $donnees = htmlspecialchars($donnees);
32         return $donnees;
33     }
34
35     $addname = validAddServicedonnees($_POST["addname"]);
36     $adddescription = validAddServicedonnees($_POST["adddescription"]);
37
38     if($addname != "" && $adddescription != "" ) {
39         $stmt = $this->pdo->prepare('INSERT INTO services (name, description)
40             VALUES (:addname, :adddescription)');
41         $stmt->bindParam(':addname', $addname);
42         $stmt->bindParam(':adddescription', $adddescription);
43         $stmt->execute();
44     } else {
45         ?>
46         <script type="text/javascript"> alert('Les informations du formulaire ne sont pas conformes') </script>
47         <?php
48     }
49 }

```

Reliés via le contrôleur :

```

263 if(isset($_POST['addname'])
264 && isset($_POST['adddescription'])) {
265     $this->serviceObject->addService(
266         ($_POST['addname']),
267         ($_POST['adddescription']));
268 }

```

Puis j'ai réalisé les mêmes éléments pour modifier les données :

```

272 if(isset($_POST['update'])){
273     $serviceById = $this->serviceObject->getServiceDetailById($_POST['update']);
274 }
275 if(isset($_POST['updatename'])
276 && isset($_POST['updatedescription'])){
277     $this->serviceObject->updateService(
278         $_POST['updateid'],
279         $_POST['updatename'],
280         $_POST['updatedescription']);
281 }

```

```

59 public function updateService(int $updateid, string $updatename, string $updatedescription)
60 {
61     function validUpdateServicedonnees($donnees) {
62         $donnees = trim($donnees);
63         $donnees = stripslashes($donnees);
64         $donnees = htmlspecialchars($donnees);
65         return $donnees;
66     }
67
68     $updatename = validAddServicedonnees($_POST["updatename"]);
69     $updatedescription = validAddServicedonnees($_POST["updatedescription"]);
70
71     $stmt = $this->pdo->prepare('UPDATE services
72         SET
73         name = :updatename,
74         description = :updatedescription
75         WHERE id = :updateid');
76     $stmt->bindParam(':updateid', $updateid);
77     $stmt->bindParam(':updatename', $updatename);
78     $stmt->bindParam(':updatedescription', $updatedescription);
79     $stmt->execute();
80 }
81

```

Et pour finir j'ai ajouté un bouton pour supprimer une ligne de données :
J'ai préféré utiliser la fonctionnalité DELETE afin de ne pas pouvoir réutiliser des identifiants déjà attribués.

```
117 <form action="" method="POST" onsubmit="return confirm('Confirmez-vous la suppression?');">
118   <button type="submit" name="deleteStaff" value="{?=$user['userid'] ?}" class="btn btn-light">
119     <svg xmlns="http://www.w3.org/2000/svg" width="16" height="16" fill="currentColor" class="bi bi-trash3" viewBox="0 0 16 16">
120       <path d="M6.5 1h3a.5.5 0 0 1 .5.5v1H6v-1a.5.5 0 0 1 .5-.5zM11 2.5v11a1.5 1.5 0 0 0 9.5 0h-3A1.5 1.5 0 0 0 15 12.5v-9.5z"/>
121     </svg>
122   </button>
123 </form>
```

```
269         if(isset($_POST['delete'])){
270             $this->serviceObject->deleteService($_POST['delete']);
271         }
```

```
51     public function deleteservice (int $id)
52     {
53         $stmt = $this->pdo->prepare('DELETE FROM services
54             WHERE id = :id');
55         $stmt->bindParam(':id', $id);
56         $stmt->execute();
57     }
```

```

140 <form action="" method="POST">
141   <div class="form-group">
142     <label for="updateid">Id : </label>
143     <input type="text" name="updateid" readonly class="form-control" id="updateid" value="<?={$_POST['update']}>" required>
144     <label for="updateName">Nom : </label>
145     <input type="text" name="updateName" class="form-control" id="updateName" value="<?={$serviceById->getServiceName()}>" required>
146     <label for="updateDescription">Description : </label>
147     <input type="text" name="updateDescription" class="form-control" id="updateDescription" value="<?={$serviceById->getServiceDescription()}>" required>
148   </div>
149   <button type="submit" class="btn backendButton" id="buttonUpdateService">Modifier</button>
150 </form>

```

Sécurité

J'ai utilisé bindParam et execute pour prévenir les injection SQL

```
97 $stmt->bindParam(':addstaffright', $addstaffright);
98 $stmt->execute();
```


J'ai sécurisé les formulaires en validant les données saisies par les utilisateurs :

```
if($addpicture1 != ""  
    && $addpicture2 != ""  
    && $addpicture3 != ""  
    && $addpicture4 != ""
```

J'ai également ajouté une fonction pour échapper les caractères spéciaux :

```
61     function validUpdateServicedonnees($donnees) {  
62         $donnees = trim($donnees);  
63         $donnees = stripslashes($donnees);  
64         $donnees = htmlspecialchars($donnees);  
65         return $donnees;  
66     }  
67  
68     $updatename = validAddServicedonnees($_POST["updatename"]);  
69     $updatedescription = validAddServicedonnees($_POST["updatedescription"]);
```

J'ai également sécurisé les mots de passe en les hachant :

```
87     $encrypted_password = password_hash($addstaffpassword, PASSWORD_BCRYPT);
```