Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE

MINISTÈRE CHARGÉ DE L'EMPLOI

DOSSIER PROFESSIONNEL (DP)

Nom de naissance	*Leliard*
Nom d'usage	*Leliard*
Prénom	*René*
Adresse	*2 rue de la Bronque Saint Victor La Coste 30290*

Titre professionnel visé

Développeur Web et Web Mobile

MODALITE D'ACCES :

- ✓ Parcours de formation
- ☐ Validation des Acquis de l'Expérience (VAE)

Présentation du dossier

Le dossier professionnel (DP) constitue un élément du système de validation du titre professionnel. **Ce titre est délivré par le Ministère chargé de l'emploi.**

Le DP appartient au candidat. Il le conserve, l'actualise durant son parcours et le présente **obligatoirement à chaque session d'examen**.

Pour rédiger le DP, le candidat peut être aidé par un formateur ou par un accompagnateur VAE.

Il est consulté par le jury au moment de la session d'examen.

Pour prendre sa décision, le jury dispose :

1. des résultats de la mise en situation professionnelle complétés, éventuellement, du questionnaire professionnel ou de l'entretien professionnel ou de l'entretien technique ou du questionnement à partir de productions.
2. du **Dossier Professionnel** (DP) dans lequel le candidat a consigné les preuves de sa pratique professionnelle.
3. des résultats des évaluations passées en cours de formation lorsque le candidat évalué est issu d'un parcours de formation
4. de l'entretien final (dans le cadre de la session titre).

[Arrêté du 22 décembre 2015, relatif aux conditions de délivrance des titres professionnels du ministère chargé de l'Emploi]

Ce dossier comporte :

- pour chaque activité-type du titre visé, un à trois exemples de pratique professionnelle ;
- un tableau à renseigner si le candidat souhaite porter à la connaissance du jury la détention d'un titre, d'un diplôme, d'un certificat de qualification professionnelle (CQP) ou des attestations de formation ;
- une déclaration sur l'honneur à compléter et à signer ;
- des documents illustrant la pratique professionnelle du candidat (facultatif)
- des annexes, si nécessaire.

Pour compléter ce dossier, le candidat dispose d'un site web en accès libre sur le site.

http://travail-emploi.gouv.fr/titres-professionnels

Sommaire

Exemples de pratique professionnelle

EXEMPLES DE PRATIQUE PROFESSIONNELLE

DOSSIER PROFESSIONNEL (DP)

Activité-type 1

Développer la partie front-end d'une application web ou web mobile sécurisée

Exemple n°1 ▸ *Mini-jeux JavaScript en ligne (HTML/CSS, Bootstrap, JavaScript)*

1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

Afin de mettre en pratique ce que j'ai appris concernant le maquettage et les langages HTML, CSS et JavaScript, j'ai créé un site de mini-jeux en ligne.

2. Précisez les moyens utilisés :

- Le zoning Mobile et Desktop de la page d'accueil :

- Le mockup Mobile et Desktop de page d'accueil :

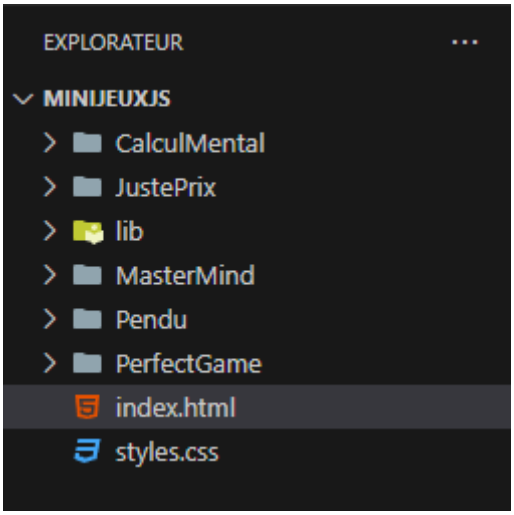
Concernant la partie développement, j'ai utilisé l'éditeur de code Visual Studio Code.

J'ai utilisé les langages HTML/CSS, Bootstrap et Javascript.

Mon site est composé d'une partie principale ainsi que de 5 parties contenant les mini-jeux.

- Un fichier index.html à la racine du projet, point d'entrée du site
- 5 dossiers contenant chacun les fichiers index.html, styles.css et script.js

- **Dossier spécifique pour chaque mini-jeu** : Chaque mini-jeu a son propre dossier contenant ses fichiers HTML, CSS et JavaScript, ce qui permet une meilleure isolation du code et évite les conflits.



- **Liens de navigation** : Les balises <a> permettent de lier la page d'accueil aux pages de chaque mini-jeu. Par exemple, le lien pour accéder au premier mini-jeu ai
Le juste prix

```
<section>
<div class="row">
  <div class="col">
    <div class="card" style="width: 18rem">
      <img
        src="JustePrix/img/juste-prix.jpg"
        class="card-img-top"
        alt="Décoration"
      />
      <div class="card-body">
        <h5 class="card-title">Jeu du juste prix.</h5>
        <p class="card-text">
          Lancez vous dans se mini-jeu ou vous devrez devinez un chiffre
          entre 0 et 1000. Dépéchez-vous, 30secondes vous sont accorder!!!
        </p>
        <a href="JustePrix/justeprix.html" class="btn btn-primary"
          >Le juste prix</a
        >
      </div>
    </div>
  </div>
</div>
```

Bien sûr, j'ai utilisé les balise header, section et footer afin de facilité la lecture du code et respecté les bonnes pratiques.

```
<body>
    <div class="container-fluid text-center">
      <header>
      <h1>Liste des mini-jeux</h1>
    </header>
    <section>
    <div class="row">
      <div class="col">
        <div class="card" style="width: 18rem">
          <img
```

```
      </div>
    </section>
    <footer>
      <p>&copy; 2024 Leliard René</p>
    </footer>
```

En ce qui concerne les jeux, je prends exemple du premier jeux ''Le juste prix''.

1. Initialisation des Variables et des Éléments DOM

- **Variables principales** :
  - NumberToFind: Le chiffre que l'utilisateur doit deviner, généré aléatoirement au début du jeu.
  - TempsRestant: Le temps restant pour le jeu, initialisé à 30 secondes.
  - compteurInterval: La référence à l'intervalle de temps qui sera utilisé pour le compte à rebours.
- **Éléments DOM** :
  - resultDiv: L'élément où le résultat du test de l'utilisateur sera affiché (ex: "C'est plus !", "C'est moins !", "C'est gagné !").
  - reboursDiv: L'élément où le temps restant est affiché.
  - GamePropalDiv: La section contenant les éléments interactifs du jeu (champ de saisie pour les propositions, bouton de validation).

```
let NumberToFind = 0;
const resultDiv = document.getElementById("resultDiv");
const reboursDiv = document.getElementById("CompteARebours");
const GamePropalDiv = document.getElementById("GamePropalDiv");
let TempsRestant = 0;
let compteurInterval = null;
```

2. Gestion des Événements

- **Démarrer le jeu** :
  - Un écouteur d'événements est placé sur le bouton avec l'ID beginGame. Lorsqu'il est cliqué, la fonction launchGame est appelée pour initialiser et démarrer le jeu.
  -
- **Vérifier une proposition** :
  - Un écouteur d'événements est placé sur le bouton avec l'ID checkPropalButton. Lorsqu'il est cliqué, la fonction checkPropal est appelée pour vérifier la proposition de l'utilisateur.
  - Un autre écouteur est placé sur le champ de saisie userPropalInput pour vérifier si l'utilisateur appuie sur la touche "Enter" pour soumettre sa proposition.

```
document.getElementById("beginGame")
    .addEventListener("click", function () {
        launchGame();
    });

document.getElementById("checkPropalButton")
    .addEventListener("click", function () {
        checkPropal();
    });

document.getElementById("userPropalInput")
    .addEventListener("keyup", function (event) {
        if (event.key == 'Enter') {
            checkPropal();
        }
    });
```

3. Fonction de Vérification de la Proposition (checkPropal)

- **Comparaison du chiffre proposé par l'utilisateur avec le chiffre à deviner** :
  - Si la proposition (numberPropal) est inférieure au chiffre à deviner (NumberToFind), le message "C'est plus !" est affiché, et un son est joué.
  - Si la proposition est supérieure, le message "C'est moins !" est affiché, et un autre son est joué.
  - Si la proposition est correcte, le message "C'est gagné !" est affiché et la fonction endGame(true) est appelée pour terminer le jeu avec succès.

```
function checkPropal() {
    let numberPropal = document.getElementById("userPropalInput").value;
    if (NumberToFind > numberPropal) {
        resultDiv.textContent = "C'est plus ! ";
        let audio = new Audio("audio/plus.mp3");
        audio.play();
    }
    else if (NumberToFind < numberPropal) {
        resultDiv.textContent = "C'est moins ! ";
        let audio = new Audio("audio/moins.mp3");
        audio.play();
    }
    else if (NumberToFind == numberPropal) {
        resultDiv.textContent = "C'est gagné ! ";
        endGame(true);
    }
}
```

4. Lancement du Jeu (launchGame)

- **Initialisation** :
  - Le jeu démarre en générant un chiffre aléatoire entre 0 et 999 (Utils.getRandomInt(1000)).
  - Le compte à rebours est initialisé à 30 secondes, et la section des propositions de jeu (GamePropalDiv) est rendue visible.
- **Compte à rebours** :
  - Un intervalle est défini pour décrémenter le temps restant chaque seconde et mettre à jour l'affichage.
  - Le style du texte de l'affichage du compte à rebours change selon le temps restant :
    - cool pour plus de 20 secondes restantes.
    - warning pour entre 10 et 20 secondes.
    - danger pour moins de 10 secondes.
  - Si le temps écoulé atteint 0, la fonction endGame(false) est appelée pour terminer le jeu avec un échec.

```
function launchGame() {
    Confetti.stopAnimationConfeti();
    NumberToFind = Utils.getRandomInt(1000);
    TempsRestant = 30;
    GamePropalDiv.style.display = "block";
    if (compteurInterval != null) {
        clearInterval(compteurInterval);
    }
    compteurInterval = setInterval(() => {
        reboursDiv.innerText = TempsRestant;
        TempsRestant--;

        if (TempsRestant >= 20) {
            reboursDiv.classList.remove("warning");
            reboursDiv.classList.remove("danger");
            reboursDiv.classList.add("cool");
        }
        else if (TempsRestant > 10) {
            reboursDiv.classList.remove("cool");
            reboursDiv.classList.remove("danger");
            reboursDiv.classList.add("warning");
        }
        else if (TempsRestant >= 0) {
            reboursDiv.classList.remove("cool");
            reboursDiv.classList.remove("warning");
            reboursDiv.classList.add("danger");
        }
        else if (TempsRestant < 0) {
            clearInterval(compteurInterval);
            endGame(false);
        }

    }, 1000);
}
```

5. Fin du Jeu (endGame)

- **Jeu gagné** :
  - Si l'utilisateur a trouvé le bon chiffre, une animation de confettis est lancée (Confetti.launchAnimationConfeti()), un son de victoire est joué, et après 5 secondes, l'animation s'arrête.
- **Jeu perdu** :
  - Si le temps est écoulé ou si le jeu est perdu pour une autre raison, un son d'échec est joué.
- **Réinitialisation** :
  - La section des propositions est cachée (GamePropalDiv.style.display = "none";), et le compte à rebours est arrêté (clearInterval(compteurInterval);).

```
function endGame(gagne) {
    if (gagne) {
        Confetti.launchAnimationConfeti();
        let audio = new Audio("audio/Bipbip.mp3");
        audio.play();
        setTimeout(() => {
            Confetti.stopAnimationConfeti();
        }, 5000);
    }
    else {
        let audio = new Audio("audio/tes_mauvais_jack.mp3");
        audio.play();
    }
    GamePropalDiv.style.display = "none";
    clearInterval(compteurInterval);

}
```

6. Bibliothèques Externes Utilisées

- **Confetti** : Gère les animations de confettis pour célébrer la victoire.
- **Utils** : Fournit des utilitaires, comme la fonction getRandomInt pour générer un chiffre aléatoire.

lib > Utils > utils.js > Utils

```
1  export class Utils{
2      static getRandomInt(max) {
3          return Math.floor(Math.random() * max);
4      }
5  }
```

```css
.confetti{
    width: 10px;
    height: 10px;
    background-color: red;
    animation: falling 5s infinite;
    position:absolute;
    top:-20px;
    z-index: 2;
}

@keyframes falling{
    0%{
        top:-20px;
        transform: rotate(180deg);
    }
    20% {
        transform: translateX(100px) rotate(360deg);
    }
    40%{
        transform: translateX(40px) rotate(180deg);
    }
    50%{
        opacity: 1;
        transform: translateX(20px) rotate(360deg);
    }
    70%{
        opacity: 0.2;
        transform: translateX(30px) rotate(180deg);
    }
    100%{
        opacity: 0;
        top:100%;
        transform: translateX(20px) rotate(360deg);
    }
}
```

```
export class Confetti{
    static launchAnimationConfeti(){
        let animateDiv = document.createElement("div");
        animateDiv.id = "allconfettis";
        animateDiv.innerHTML = "";

        for(let i =0; i < 100; i++){
            let confeti = document.createElement("div");
            confeti.classList.add("confetti");
            confeti.style.left = this.getRandomArbitrary(0,100)+'%';
            confeti.style.animationDelay = 50*i+"ms";
            confeti.style.backgroundColor = '#'+(Math.random()*0xFFFFFF<<0).toString(16);
            animateDiv.appendChild(confeti);
        }

        document.body.appendChild(animateDiv);
    }


    static stopAnimationConfeti(){
        let animateDiv = document.getElementById("allconfettis");
        if(animateDiv != undefined){
            animateDiv.innerHTML = "";
            animateDiv.remove();
        }
    }

    static getRandomArbitrary(min, max) {
        return Math.floor(Math.random() * (max - min) + min);
    }
}
```

3. Avec qui avez-vous travaillé ?

J'ai travaillé seul.

4. Contexte

Nom de l'entreprise, organisme ou association ▶ Studi

Chantier, atelier, service ▸

Période d'exercice ▸ Du : 06/05/2024 au : 20/05/2024

5. Informations complémentaires *(facultatif)*

Améliorations Potentielles

- **Gestion des erreurs** : Ajouter une vérification pour s'assurer que l'utilisateur entre bien un nombre valide.
- **Réinitialisation complète** : Offrir la possibilité de redémarrer le jeu sans rafraîchir la page.
- **UI/UX** : Ajouter des animations ou des messages pour guider l'utilisateur pendant le jeu.

Activité-type 1 — Développer la partie front-end d'une application web ou web mobile sécurisée

Exemple n°2 ▸ *Quai antique (Figma, HTML/SCSS, Javascript)*

1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

Projet : Réalisation de la partie front-end pour le site d'un restaurant.

Objectif : Développer une interface utilisateur (UI) responsive, moderne et sécurisée.

Outils utilisés :

- **Figma** : Création des maquettes et de la charte graphique.
- **HTML/CSS avec Bootstrap et SaSS** : Mise en place de la structure et du style.
- **JavaScript** : Dynamisation des pages web (ex: interactions utilisateurs)

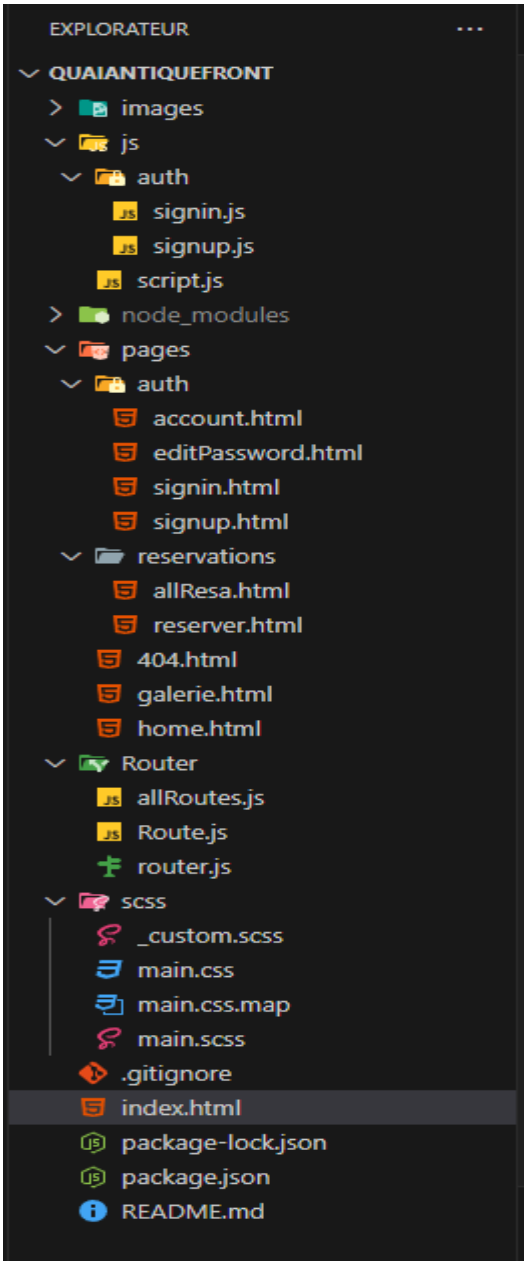
Conception des maquettes :

- Utilisation de Figma pour élaborer des maquettes respectant les standards UX/UI.

- Travail sur une charte graphique cohérente pour représenter l'identité visuelle du restaurant.
- Annexe en page 48

Développement en HTML/CSS avec Bootstrap :

- Structure des dossiers :



images/ : Contient les ressources visuelles.

js/ : Dossier pour les fichiers JavaScript :

- **auth/** : Gestion de l'authentification (signin.js, signup.js).
- **Router/** : Fichiers pour gérer la navigation (allRoutes.js, router.js).

pages/ : Dossier avec les pages HTML :

- **auth/** : Pages liées à l'authentification (connexion, inscription).
- **reservations/** : Pages de gestion des réservations.

scss/ : Contient les fichiers SCSS et CSS pour le style.

index.html : Fichier principal du site.

En-tête (Header) :

- **Responsive** grâce à l'utilisation de Bootstrap.

```html
<!DOCTYPE html>
<html lang="fr-FR">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
    <link href="https://fonts.googleapis.com/css2?family=Hind+Madurai:wght@300;400;500;600;700&display=swap" rel="stylesheet">
    <link rel="stylesheet" href="scss/main.css">
    <title>Quai Antique</title>
</head>
<body>
```

Utilisation du fichier scss.main.css pour importer Bootstrap ainsi que pour le style.

```scss
@import url("/node_modules/bootstrap-icons/font/bootstrap-icons.css");
@import 'custom';

html {
    position: relative;
    min-height: 100%;
}
```

Le fichier custom.scss modifie et étend le framework Bootstrap en intégrant des variables personnalisées pour les couleurs et les polices de caractères :

1. Variables personnalisées :

- $primary: Couleur principale (#906427).
- $secondary: Couleur secondaire (#B6AC97).
- $dark: Couleur pour les éléments sombres (#392C1E).
- $black: Couleur pour le noir (#292222).
- $font-family-sans-serif: "Montserrat" pour les éléments sans-serif.
- $font-family-serif: "Hind Madurai" pour les éléments serif.

2. Importation de Bootstrap :

- Après avoir défini ces variables, le fichier **Bootstrap SCSS** est importé. Bootstrap utilisera ces nouvelles variables pour générer les styles.

```scss
$primary: #906427;
$secondary: #B6AC97;
$dark: #392C1E;
$black: #292222;

$font-family-sans-serif: "Monserrat", sans-serif !default;
$font-family-serif: "Hind Madurai", serif !default;

@import "../node_modules/bootstrap/scss/bootstrap";
```

- **Barre de navigation** avec des liens vers les différentes sections : Accueil, Galerie, La Carte, Réservations, Mon Compte, Connexion/Déconnexion.

```
<header>
    <nav class="navbar navbar-expand-lg bg-dark" data-bs-theme="dark">
        <div class="container-fluid">
        <a class="navbar-brand" href="/">Quai Antique</a>
        <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent"
        aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">
            <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse" id="navbarSupportedContent">
            <ul class="navbar-nav ms-auto mb-2 mb-lg-0">
            <li class="nav-item">
                <a class="nav-link" href="/">Accueil</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="/galerie">Galerie</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="#">La carte</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="/allResa">Les réservations</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="/account">Mon compte</a>
            </li>
            <li class="nav-item" data-show="disconnected">
                <a class="nav-link" href="/signin">Connexion</a>
            </li>
            <li class="nav-item" data-show="connected">
                <button class="nav-link" id="BtnSignout">Déconnexion</button>
            </li>
            </ul>
        </div>
        </div>
    </nav>
</header>
```

Voici un détail des étapes dans mon code de validation et inscription des utilisateurs, allant de la déclaration des variables jusqu'à l'appel API :

1. Déclaration des variables

- Champs d'entrée : Déclaration des variables pour chaque champ duformulaire (nom, prénom, email, mot de passe) en utilisant document.getElementById() pour cibler chaque élément.
- Bouton de validation : La variable btnValidation est liée au bouton de soumission du formulaire.

```
3   const inputNom = document.getElementById("NomInput");
4   const inputPrenom = document.getElementById("PrenomInput");
5   const inputMail = document.getElementById("EmailInput");
6   const inputPassword = document.getElementById("PasswordInput");
7   const inputValidatePassword = document.getElementById("ValidatePasswordInput");
8   const btnValidation = document.getElementById("btn-validation-inscription");
9   const formInscription = document.getElementById("formulaireInscription");
```

- Formulaire : formInscription est lié à l'élément <form> pour regrouper les données.

2. Ajout des écouteurs d'événements

- Chaque champ d'entrée a un écouteur keyup qui déclenche la fonction validateForm pour valider les données en temps réel.
- Le bouton de validation a un écouteur d'événement click, qui appelle la fonction inscrireUtilisateur lorsque l'utilisateur clique dessus.

```
inputNom.addEventListener("keyup", validateForm);
inputPrenom.addEventListener("keyup", validateForm);
inputMail.addEventListener("keyup", validateForm);
inputPassword.addEventListener("keyup", validateForm);
inputValidatePassword.addEventListener("keyup", validateForm);
btnValidation.addEventListener("click", inscrireUtilisateur);
```

3. Validation des formulaires

- validateForm() : Cette fonction valide chaque champ :
    - Vérification de la présence des données avec validateRequired().
    - Validation de l'email avec une regex dans validateMail().
    - Validation du mot de passe avec une regex dans validatePassword().

- Confirmation que les mots de passe correspondent avec validateConfirmationPassword().

- Le bouton de soumission est activé ou désactivé en fonction de la validité des champs.

```
function validateForm() {
    const nomOk = validateRequired(inputNom,);
    const prenomOk = validateRequired(inputPrenom,);
    const mailOk = validateMail(inputMail);
    const passwordOk = validatePassword(inputPassword);
    const passwordConfirmOk = validateConfirmationPassword(inputPassword, inputValidatePassword)

    if (nomOk && prenomOk && mailOk && passwordOk && passwordConfirmOk) {
        btnValidation.disabled = false;
    } else {
        btnValidation.disabled = true;
    }
}
```

5. Soumission du formulaire et appel API

- Dans la fonction inscrireUtilisateur() :
    - Les données du formulaire sont récupérées avec FormData et stockées dans un objet JSON (raw).
    - Une requête POST est envoyée à l'API à l'aide de fetch(), en spécifiant l'URL de l'API, les en-têtes, et le corps de la requête (raw).
    - Si l'appel est réussi, l'utilisateur est redirigé vers la page de connexion (/signin), sinon, un message d'erreur est affiché.

```javascript
function inscrireUtilisateur() {
    let dataForm = new FormData(formInscription);

    let myHeaders = new Headers();
    myHeaders.append("Content-Type", "application/json");

    let raw = JSON.stringify({
        "firstName": dataForm.get("nom"),
        "lastName": dataForm.get("prenom"),
        "email": dataForm.get("email"),
        "password": dataForm.get("mdp")
    });

    let requestOptions = {
        method: "POST",
        headers: myHeaders,
        body: raw,
        redirect: "follow"
    };

    fetch(apiUrl + "registration", requestOptions)
        .then(response => {
            if (response.ok) {
                return response.json();
            } else {
                alert("Erreur lors de l'inscription");
            }
        })
        .then(result => {
            alert("Bravo," + dataForm.get("prenom") + ", vous êtes maintenant inscrit, vous pouvez vous connecter")
            document.location.href = "/signin";
            console.log(result);
        })
        .catch((error) => console.error(error));
}
```

Chaque étape assure que les données sont validées et formatées avant d'être envoyées à l'API, garantissant ainsi un traitement sécurisé et correct des informations.

Annexes page 48

Activité-type 2

Développer la partie back-end d'une application web ou web mobile sécurisée

Exemple n° 1 ▸ *Face-Twit*

1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

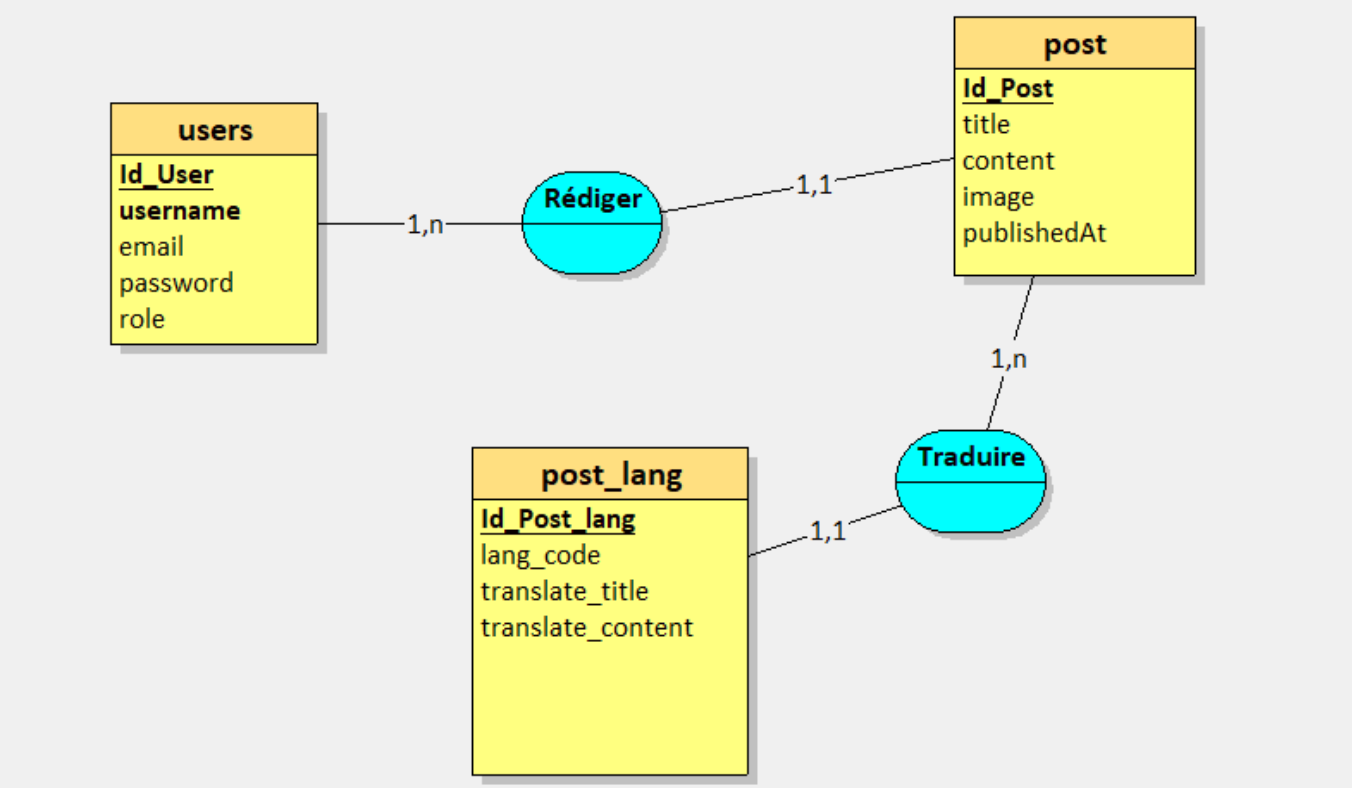
Objectifs du projet :

- Créer un mini-réseau social complet avec **Symfony**, en respectant les principes de MVC (Modèle-Vue-Contrôleur).
- Utiliser **MongoDB** pour la gestion spécifique des interactions telles que les likes.

2. Précisez les moyens utilisés :

Conception :

- **Looping** pour la méthode Merise



Un Users peut rédiger plusieurs Post et un Post ne peut être rédiger que part un seul Users.

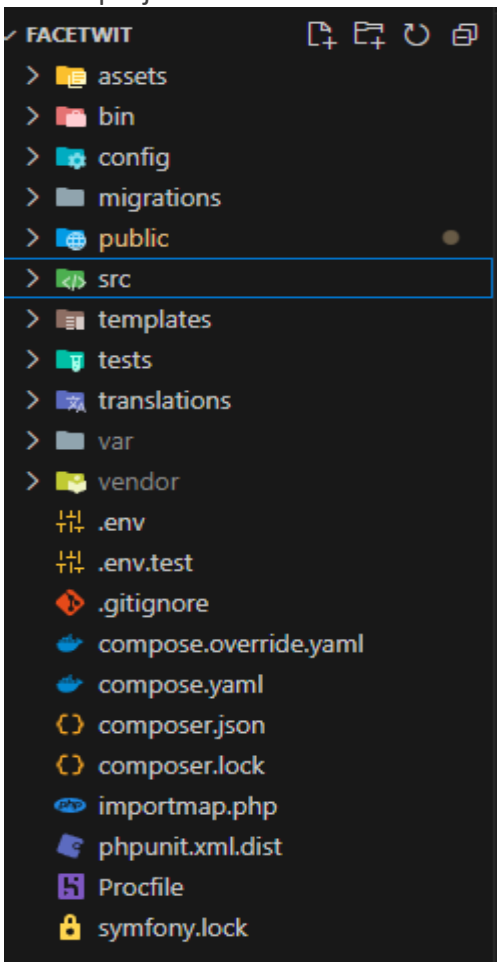
Un Post peut être traduit en plusieurs langues et une langue ne peu traduire qu'un seul Post.

Le script MySQL de création de BDD et des tables **:**

Annex page : 48

Réalisation :

- **Symfony** pour le développement du projet MVC.



- **Doctrine MongoDB ODM** (Object Document Mapper) pour la gestion des données non relationnelles comme les likes.

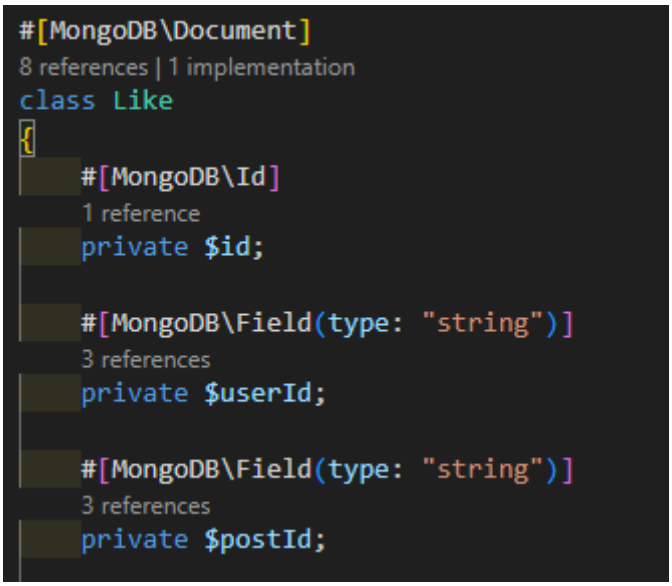
```
namespace App\Document;

use Doctrine\ODM\MongoDB\Mapping\Annotations as MongoDB;
```

Attributs de la classe :

- **$id** : Identifiant unique du like, généré automatiquement par MongoDB.
- **$userId** : ID de l'utilisateur qui a aimé le post.
- **$postId** : ID du post qui a été aimé.

Annotations MongoDB :

- #[MongoDB\Document] indique que cette classe est un document MongoDB.
- #[MongoDB\Field(type: "string")] spécifie que les champs sont de type chaîne de caractères.



```
#[MongoDB\Document]
8 references | 1 implementation
class Like
{
    #[MongoDB\Id]
    1 reference
    private $id;

    #[MongoDB\Field(type: "string")]
    3 references
    private $userId;

    #[MongoDB\Field(type: "string")]
    3 references
    private $postId;
```

Un script JS afin de gérer les actions de "like" et "unlike" sur des publications du réseau social.

1. **Chargement du DOM :**

- Le script est encapsulé dans un écouteur d'événement DOMContentLoaded, ce qui garantit que le code est exécuté uniquement lorsque toute la structure HTML est chargée.
- Cela permet d'éviter d'attacher des événements à des éléments qui ne sont pas encore chargés

```
document.addEventListener('DOMContentLoaded', () => {
```

2. Ajout d'événements pour les boutons "like" :

- **Sélection des boutons** : Le script sélectionne tous les éléments qui ont la classe .like-button.
- **Écouteur d'événement "click"** : Pour chaque bouton "like", un écouteur d'événement est ajouté. Lorsqu'un bouton est cliqué, l'action de "like" est déclenchée.
- **Récupération des attributs HTML personnalisés** : Le script extrait les attributs data-post-id, data-user-id, et data-csrf-token des boutons. Ces informations sont nécessaires pour faire la requête.

```
    document.querySelectorAll('.like-button').forEach(button => {
        button.addEventListener('click', () => {
            const postId = button.getAttribute('data-post-id');
            const userId = button.getAttribute('data-user-id');
            const csrfToken = button.getAttribute('data-csrf-token');
```

3. Envoi de la requête fetch pour "like" :

- **Requête HTTP POST** : La méthode fetch() est utilisée pour envoyer une requête POST à l'URL /like du serveur.
- **En-têtes** :
  - Content-Type: application/json pour indiquer que le corps de la requête est au format JSON.
  - X-Requested-With: XMLHttpRequest est souvent utilisé pour indiquer une requête AJAX.
  - X-CSRF-Token pour inclure le token CSRF nécessaire à la validation côté serveur.
- **Corps de la requête** : Le corps de la requête envoie un objet JSON contenant user_id et post_id.

```
            fetch(`/like`, {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json',
                    'X-Requested-With': 'XMLHttpRequest',
                    'X-CSRF-Token': csrfToken
                },
                body: JSON.stringify({
                    user_id: userId,
                    post_id: postId
                })
            })
```

4. Gestion de la réponse serveur :

- **Traitement de la réponse** : Une fois la réponse reçue du serveur, le script tente de convertir la réponse en JSON.
- **Vérification du succès** : Le script vérifie si la réponse contient le message de succès. Si ce n'est pas le cas, une erreur est générée.
- **Mise à jour de l'interface utilisateur** : Si l'opération réussit, un message d'alerte est affiché et la page est rechargée avec location.reload().

```
21          .then(response => response.json())
22          .then(data => {
23              if (!data.message.includes('successfully')) {
24                  throw new Error(data.message);
25              }
26              alert(data.message);
27              location.reload();
28          })
```

5. Gestion des erreurs :

- Si une erreur survient (exemple : la requête échoue, ou l'utilisateur a déjà liké), elle est capturée par catch(), et un message d'erreur est affiché.

```
29          .catch(error => {
30              console.error('Error:', error);
31              alert('An error occurred: ' + error.message);
32          });
33      });
34  });
```

6. Ajout d'événements pour les boutons "unlike" :

- Cette partie est similaire à celle des "likes", mais elle concerne la requête de "unlike". Lorsque le bouton "unlike" est cliqué, une requête POST est envoyée à /unlike, et si elle réussit, la page est rechargée pour mettre à jour l'état de l'interface utilisateur.

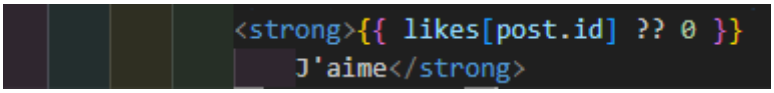
7. Résumé des points importants :

- **Sécurité** : Le script utilise un token CSRF pour protéger l'application contre les attaques CSRF.
- **Interaction dynamique** : Le script utilise fetch() pour envoyer des requêtes asynchrones, permettant de mettre à jour l'état des "likes" et "unlikes" sans recharger la page.
- **Traitement des erreurs** : Les erreurs sont gérées et affichées à l'utilisateur pour lui permettre de comprendre ce qui s'est mal passé.

Mise en place de la fonctionnalité dans le template Twig

1. **Affichage du nombre de "J'aime"** :

- Cette ligne affiche le nombre de "likes" pour un post spécifique.
- likes[post.id] récupère le nombre de likes pour le post avec l'ID correspondant.
- ?? 0 est un opérateur de coalescence null en Twig qui retourne 0 si aucune valeur n'est trouvée pour ce post



2. **Vérification si l'utilisateur est connecté** :

- Ce bloc vérifie si l'utilisateur est connecté (app.user représente l'utilisateur connecté dans Symfony). Si c'est le cas, les boutons "J'aime" et "Je n'aime plus" sont affichés.

3. **Bouton "J'aime"** :

- **Classe du bouton** : btn btn-primary applique les styles Bootstrap pour un bouton bleu.
- **data-post-id** : Attribut personnalisé pour stocker l'ID du post.
- **data-user-id** : Attribut personnalisé pour stocker l'ID de l'utilisateur connecté.
- **data-csrf-token** : Un token CSRF est généré pour sécuriser la requête associée à l'action "like". La fonction csrf_token('like') génère ce token pour le formulaire.
- Semblable au bouton "J'aime", ce bouton permet à l'utilisateur de retirer son like.
- **Classe du bouton** : btn btn-secondary applique les styles Bootstrap pour un bouton gris.

```
{% if app.user %}
    <button class="btn btn-primary like-button" data-post-id="{{ post.id }}" data-user-id="{{ app.user.id }}" data-csrf-token="{{ csrf_token('like') }}">
        J'aime
    </button>
    <button class="btn btn-secondary unlike-button" data-post-id="{{ post.id }}" data-user-id="{{ app.user.id }}" data-csrf-token="{{ csrf_token('unlike') }}">
        Je n'aime plus
    </button>
{% endif %}
```

5. Intégration avec JavaScript :

Les attributs data-post-id, data-user-id et data-csrf-token dans les boutons sont utilisés par le script JavaScript pour envoyer des requêtes AJAX lorsqu'un utilisateur clique sur "J'aime" ou "Je n'aime plus". Ces attributs fournissent les informations nécessaires pour identifier l'utilisateur, le post et pour sécuriser la requête avec un token CSRF.

6. Sécurité avec CSRF Token :

Le token CSRF est utilisé pour protéger contre les attaques de type **Cross-Site Request Forgery**. Il s'assure que chaque requête est légitime en générant un token unique pour chaque action (like/unlike).

Utilisation dans le template Twig :

Ce code est inséré dans une boucle qui parcourt les posts (post.id). Ainsi, chaque post aura ses propres boutons "J'aime" et "Je n'aime plus", ainsi que son propre nombre de likes.

Résumé :

- Le nombre de "J'aime" est affiché dynamiquement.
- Les boutons "J'aime" et "Je n'aime plus" sont disponibles uniquement pour les utilisateurs connectés.
- Chaque bouton inclut des attributs personnalisés pour l'ID du post et de l'utilisateur, ainsi qu'un token CSRF pour sécuriser l'action

Controller pour gérer les likes :

- Utilisation de **MongoDB** pour stocker les interactions de type likes, avec une performance optimisée pour les grandes quantités de données.

1. Attributs et Constructeur :

- Le constructeur injecte des services comme le **CsrfTokenManager**, les **repositories d'utilisateurs et de posts**, et un **logger** pour enregistrer les événements.

```php
2 references | 0 overrides
public function __construct(
    CsrfTokenManagerInterface $csrfTokenManager,
    UserRepository $userRepository,
    PostRepository $postRepository,
    LoggerInterface $logger
) {
    $this->csrfTokenManager = $csrfTokenManager;
    $this->userRepository = $userRepository;
    $this->postRepository = $postRepository;
    $this->logger = $logger;
}
```

2. Fonctionnalité "like" :

- Reçoit une requête, valide l'intégrité (CSRF, User ID, Post ID) avec la fonction validateRequest, et vérifie si le like existe déjà. Si tout est correct, il enregistre le like via MongoDB.

```php
#[Route(path: '/like', name: 'like_post', methods: ['POST'])]
8 references | 0 overrides
public function like(Request $request, DocumentManager $dm, LikeRepository $likeRepository): JsonResponse
{
    $content = json_decode(json: $request->getContent(), associative: true);
    $userId = $content['user_id'] ?? null;
    $postId = $content['post_id'] ?? null;

    $this->logger->info(message: 'Received like request', context: [
        'user_id' => $userId,
        'post_id' => $postId,
        'csrf_token' => $request->headers->get(key: 'X-CSRF-Token'),
    ]);

    $validationResult = $this->validateRequest(request: $request, tokenId: 'like', userId: $userId, postId: $postId);
    if ($validationResult !== null) {
        return $validationResult;
    }

    if ($this->alreadyLiked(userId: $userId, postId: $postId, likeRepository: $likeRepository)) {
        return new JsonResponse(data: ['message' => 'Already liked'], status: 400);
    }

    $like = new Like();
    $like->setUserId(userId: $userId);
    $like->setPostId(postId: $postId);

    $dm->persist(object: $like);
    $dm->flush();

    return new JsonResponse(data: ['message' => 'Post liked successfully'], status: 200);
}
```

3. Fonctionnalité "unlike" :

- Supprime un like existant si trouvé dans la base de données MongoDB.

```php
#[Route(path: '/unlike', name: 'unlike_post', methods: ['POST'])]
8 references | 0 overrides
public function unlike(Request $request, DocumentManager $dm, LikeRepository $likeRepository): JsonResponse
{
    $content = json_decode(json: $request->getContent(), associative: true);
    $userId = $content['user_id'] ?? null;
    $postId = $content['post_id'] ?? null;

    // Log the received data
    $this->logger->info(message: 'Received unlike request', context: [
        'user_id' => $userId,
        'post_id' => $postId,
        'csrf_token' => $request->headers->get(key: 'X-CSRF-Token'),
    ]);

    $validationResult = $this->validateRequest(request: $request, tokenId: 'unlike', userId: $userId, postId: $postId);
    if ($validationResult !== null) {
        return $validationResult;
    }

    $like = $likeRepository->findByUserAndPost(userId: $userId, postId: $postId);
    if (!$like) {
        return new JsonResponse(data: ['message' => 'Like not found'], status: 400);
    }

    $dm->remove(object: $like);
    $dm->flush();

    return new JsonResponse(data: ['message' => 'Post unliked successfully'], status: 200);
}
```

4. Comptage des likes :

- Renvoie le nombre total de likes pour un post spécifique.

```php
#[Route(path: '/likes/count', name: 'count_likes', methods: ['GET'])]
8 references | 0 overrides
public function countLikes(Request $request, LikeRepository $likeRepository): JsonResponse
{
    $postId = $request->get(key: 'post_id');
    if (empty($postId)) {
        return new JsonResponse(data: ['message' => 'Post ID is required'], status: 400);
    }

    $count = $likeRepository->countLikesForPost(postId: $postId);

    return new JsonResponse(data: ['count' => $count], status: 200);
}
```

5. Validation des requêtes :

- Vérification des tokens CSRF et des IDs avec des messages d'erreurs détaillés.

```php
private function validateRequest(Request $request, string $tokenId, ?string $userId, ?string $postId): ?JsonResponse
{
    $invalidToken = !$this->csrfTokenManager->isTokenValid
        (token: new CsrfToken
        (id: $tokenId, value: $request->headers->get(key: 'X-CSRF-Token')));
    $missingIds = empty($userId) || empty($postId);

    $this->logger->info(message: 'Validating request', context: [
        'invalid_token' => $invalidToken,
        'missing_ids' => $missingIds,
        'user_id' => $userId,
        'post_id' => $postId,
    ]);

    if ($invalidToken || $missingIds) {
        $message = $invalidToken ? 'Invalid CSRF token' : 'User ID and Post ID are required';
        return new JsonResponse(data: ['message' => $message], status: 400);
    }

    $user = $this->userRepository->find(id: $userId);
    $post = $this->postRepository->find(id: $postId);

    $this->logger->info(message: 'Checking user and post existence', context: [
        'user_exists' => (bool) $user,
        'post_exists' => (bool) $post,
    ]);

    if (!$user || !$post) {
        return new JsonResponse(data: ['message' => 'User or Post not found'], status: 404);
    }

    return null;
}
```

```php
    1 reference
    private function alreadyLiked(string $userId, string $postId, LikeRepository $likeRepository): bool
    {
        return (bool) $likeRepository->findByUserAndPost(userId: $userId, postId: $postId);
    }
}
```

- Les **getters** et **setters** permettent de récupérer et de modifier les valeurs des attributs $userId et $postId, respectivement.

```php
1 reference | 0 overrides
public function setUserId(string $userId): self
{
    $this->userId = $userId;

    return $this;
}

0 references | 0 overrides
public function getPostId(): ?string
{
    return $this->postId;
}
```

- Mise en place des entités et relations traditionnelles dans **MySQL** pour les utilisateurs et les posts.

```php
namespace App\Entity;

use App\Repository\UserRepository;
use Doctrine\Common\Collections\ArrayCollection;
use Doctrine\ORM\Mapping as ORM;
use Symfony\Bridge\Doctrine\Validator\Constraints\UniqueEntity;
use Symfony\Component\PasswordHasher\Hasher\UserPasswordHasherInterface;
use Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface;
use Symfony\Component\Security\Core\User\UserInterface;
use Symfony\Component\Validator\Constraints as Assert;
```

1. Propriétés de l'utilisateur :

- **id** : Clé primaire auto-générée.
- **username** : Nom d'utilisateur unique, avec validation de longueur et d'unicité.
- **roles** : Tableau de rôles, avec une valeur par défaut ROLE_USER.
- **password** : Mot de passe haché, validé par Symfony (longueur minimale de 5 caractères).

```php
#[ORM\Id]
#[ORM\GeneratedValue]
#[ORM\Column]
1 reference
private ?int $id = null;
```

```php
#[ORM\Column(length: 180)]
#[Assert\Length(
    min: 2,
    max: 180,
    minMessage: "Le nom de d'utilisateur doit contenir au moins de {{ limit }} caractères",
    maxMessage: "Le nom de d'utilisateur ne pas doit contenir plus de {{ limit }} caractères",
)]
3 references
private ?string $username = null;
```

```php
/**
 * @var list<string> The user roles
 */
#[ORM\Column]
2 references
private array $roles = ["ROLE_USER"];
```

```php
/**
 * @var string The hashed password
 */
#[ORM\Column]
#[Assert\NotBlank]
#[Assert\Length(
    min: 5,
    minMessage: "Le mot de passe doit contenir au moins {{ limit }} caractères"
)]
2 references
private ?string $password = null;

2 references
private ?string $confirm = null;
2 references
private UserPasswordHasherInterface $passwordHasher;
```

4. Méthodes principales :

- **get/setUsername** : Gère le nom d'utilisateur.
- **get/setPassword** : Gère le mot de passe (hachage inclus).
- **get/setRoles** : Gère les rôles utilisateurs.
- **get/setPosts** : Gère la relation avec les posts.

```php
0 references | 0 overrides
public function getId(): ?int
{
    return $this->id;
}

0 references | 0 overrides
public function getUsername(): ?string
{
    return $this->username;
}

2 references | 0 overrides
public function setUsername(string $username): static
{
    $this->username = $username;

    return $this;
}
```

2. Relations :

- **posts** : Relation OneToMany avec l'entité Post, permettant à un utilisateur d'avoir plusieurs posts.

```php
#[ORM\OneToMany(targetEntity: "App\Entity\Post", mappedBy: "user")]
3 references
private $posts;
```

3. Interfaces implémentées :

- **UserInterface** : Pour la gestion de l'utilisateur dans Symfony.
- **PasswordAuthenticatedUserInterface** : Gère le hachage du mot de passe avec le composant UserPasswordHasherInterface.

- **Twig** pour le rendu des vues.

Formulaire de connexion.

```twig
{% extends "base.html.twig" %}

{% block body %}
    {{ form_start(form, {'attr': {'class': 'container'}}) }}
    {{form_row(form.username)}}

    <div class="input-group mb-3">
        {{form_widget(form.password)}}
    </div>
    {{form_errors(form.password)}}
    {{ form_rest(form) }}
    <input class="btn btn-success mt-2" type="submit" value="Créer le compte">
    <div>
        <a href="/login" class="btn btn-link">Se connecter</a>
    </div>


{% endblock %}
```

- Création de formulaires Symfony pour la soumission des posts et gestion des interactions (like, commentaires).

```php
0 references | 0 overrides
public function buildForm(FormBuilderInterface $builder, array $options): void
{
    $builder
        ->add(child: "title", type: TextType::class, options: [
            "label" => "Titre",
            "required" => true,
        ])
        ->add(child: "content", type: TextareaType::class, options: [
            "label" => "Contenu",
            "required" => true,
        ])
        ->add(child: "image", type: FileType::class, options: [
            "label" => "L'image",
            'mapped' => false,
            "required" => false,
            'constraints' => [
                new File(options: [
                    'maxSize' => '15M',
                    'mimeTypes' => [
                        'image/jpeg',
                        "image/gif",
                        "image/png",
                        "image/svg+xml",
                        "image/jpg",
                        "image/webp"
                    ],
                    'mimeTypesMessage' => 'Veuillez proposer une image valide.',
                ])
            ],
        ]);
}
```

Résultats obtenus :

- Mise en place d'un réseau social fonctionnel, avec une gestion efficace des likes via MongoDB, ce qui permet une grande scalabilité.
- Utilisation d'une architecture hybride (relationnelle pour utilisateurs/posts et documentaire pour les interactions).
- Sécurisation des interactions utilisateur avec validation et authentification via Symfony.

Ce projet montre la capacité à intégrer des bases de données relationnelles (MySQL) et non relationnelles (MongoDB) dans une application Symfony complète, tout en respectant les bonnes pratiques de sécurité et d'optimisation.

3. Avec qui avez-vous travaillé ?

J'ai travailler seul.

4. Contexte

Nom de l'entreprise, organisme ou association ▸

Chantier, atelier, service ▸

Période d'exercice ▸ Du : 01/04 au : 15/04

5. Informations complémentaires *(facultatif)*

Activité-type 2

Développer la partie back-end d'une application web ou web mobile sécurisée

Exemple n° 1 ▸ *Conception d'une base d données (MySQL)*

1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

Au cours de la formation, nous avons abordé la conception et le développement d'une base de données à l'aide de la méthode Merise.

Pour mettre en application ce que nous avions appris, je me suis inspiré d'une évaluation nommé « Garage ».

Au sein de cette application, il existe plusieurs voitures, qui contiennent plusieurs options.

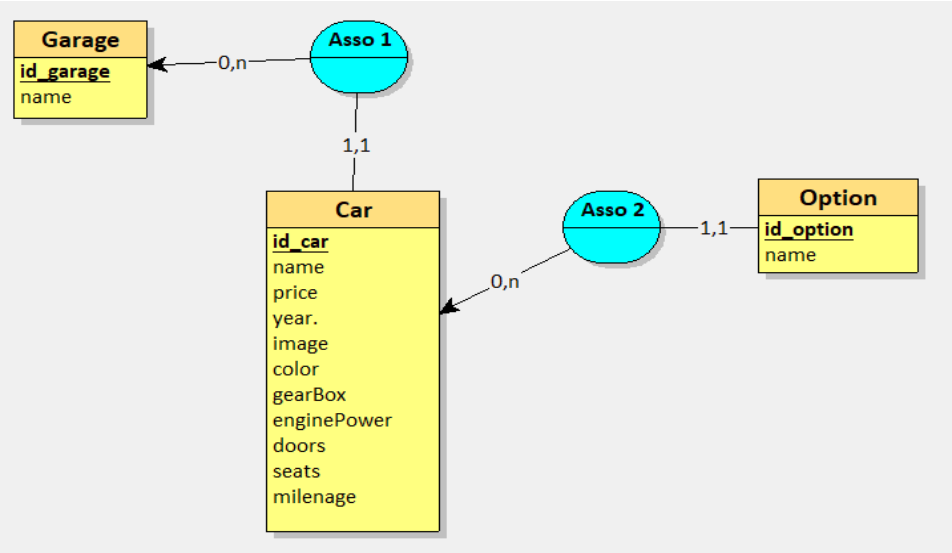
2. Précisez les moyens utilisés :

J'ai commencé par élaborer le MCD, le modèle conceptuel de données, qui consiste à créer les différentes entités qui composeront le système d'information, et à établir des relations entre-elles.

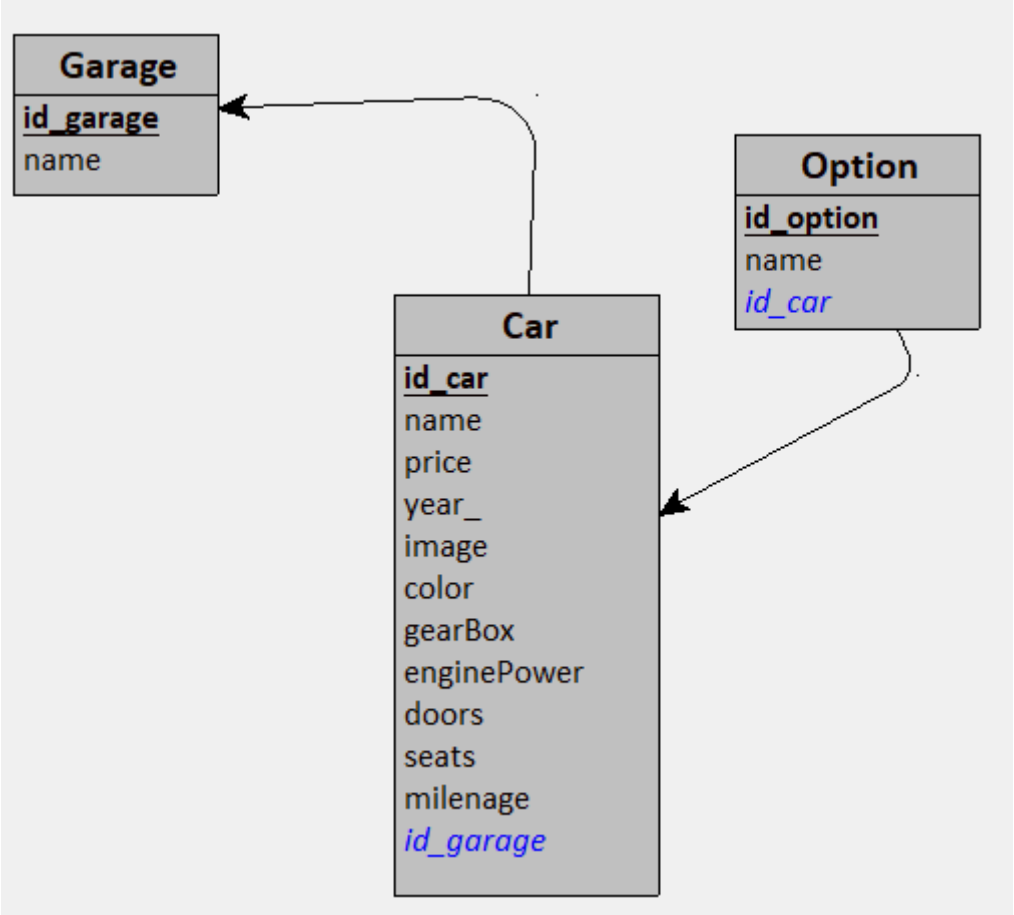
Ainsi, j'obtiens les entités « Garage », « Voiture » et « Option », composée de leur propriétés.
Ensuite, j'établis les cardinalités :

- ➔ un garage possède 0 ou plusieurs voitures et un voiture appartient à un et un seul garage.
- ➔ une voiture est équipée de 0 ou plusieurs options et un option équipe une et une seule voiture.

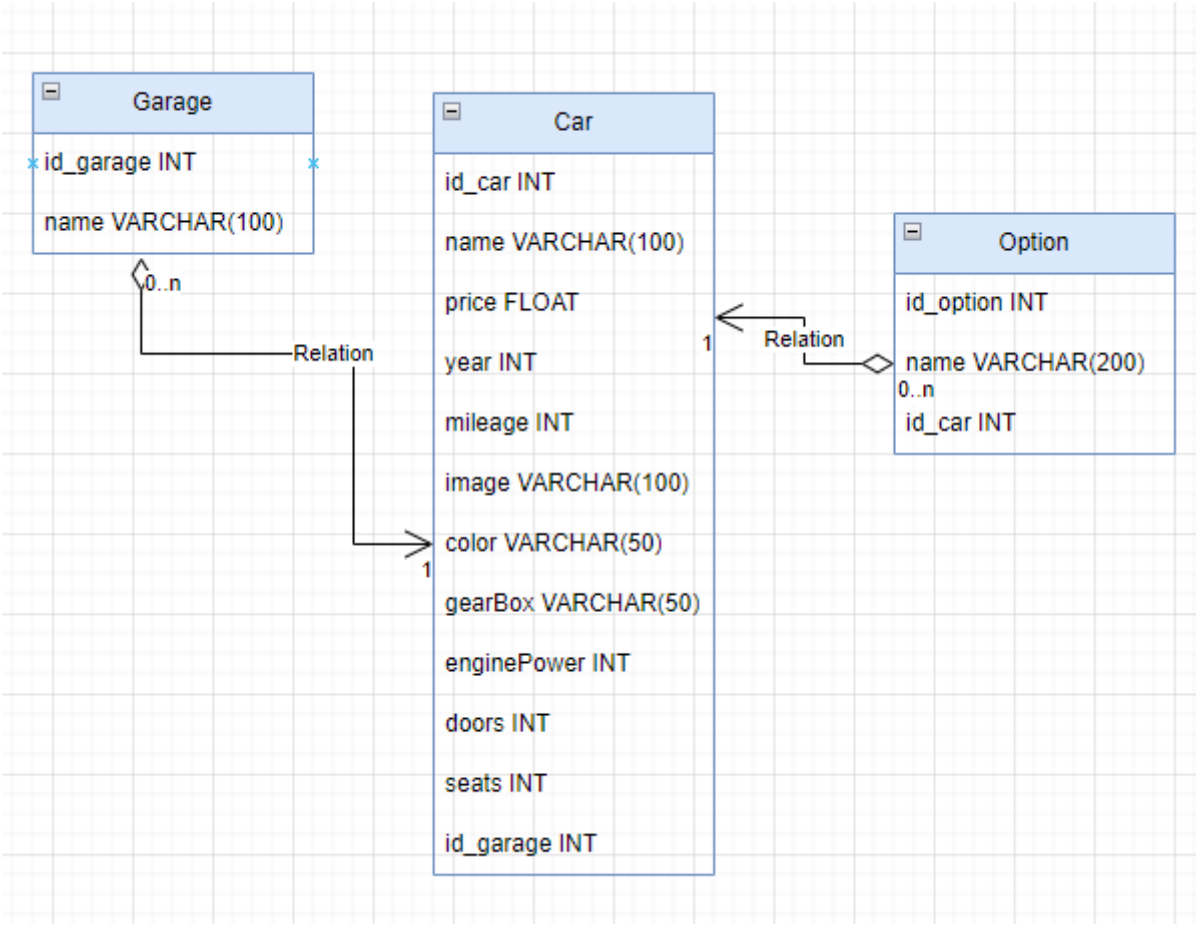
D'où le MCD (Modèle Conceptuel de Données) suivant :



Pour les relations (1,1 0,n), la clé primaire de la table côté (0,n) devient une clé étrangère de la table côté (1,1) créant ainsi la relation entre les tables :



Le MPD (Modèle Physique de Données) est donc le suivant :



UNE FOIS LE MPD ETABLI, JE PEUX ECRIRE LES INSTRUCTIONS SQL QUI PERMETTRA DE GENERER LA BASE DE DONNEES :

```sql
CREATE DATABASE IF NOT EXISTS garages;

USE garages;

CREATE TABLE IF NOT EXISTS garage (
    id_garage INT AUTO_INCREMENT NOT NULL,
    name_ VARCHAR(50) NOT NULL,
    PRIMARY KEY(id_garage)
);

CREATE TABLE IF NOT EXISTS car(
    id_car INT AUTO_INCREMENT NOT NULL,
    name_ VARCHAR(100) NOT NULL,
    price DECIMAL(15,2) NOT NULL,
    year_ INT NOT NULL,
    image_ VARCHAR(100) NOT NULL,
    color VARCHAR(50) NOT NULL,
    gearBox VARCHAR(50) NOT NULL,
    enginePower INT NOT NULL,
    doors INT NOT NULL,
    seats INT NOT NULL,
    milenage INT NOT NULL,
    id_garage INT NOT NULL,
    PRIMARY KEY(id_car),
    FOREIGN KEY(id_garage) REFERENCES Garage(id_garage)
);

CREATE TABLE IF NOT EXISTS 'option'(
    id_option INT AUTO_INCREMENT NOT NULL,
    name_ VARCHAR(200) NOT NULL,
    id_car INT NOT NULL,
    PRIMARY KEY(id_option),
    FOREIGN KEY(id_car) REFERENCES Car(id_car)
);
```

J'AI VOLONTAIREMENT OMIS D'INDIQUER 'ON DELETE CASCADE' SUR LES CLES ETRANGERES POUR POUVOIR CREER ET TESTER UN TRIGGER QUI SIMULERA L'INSTRUCTION 'ON DELETE CASCADE':

```sql
DELIMITTER |
CREATE TRIGGER IF NOT EXISTS trigger_delete_car AFTER DELETE on car
FOR EACH ROW
BEGIN
DELETE FROM 'option'
    WHERE id_car = OLD.id_car;
END
```

AINSI, LORSQUE JE SUPPRIME UN ENREGISTREMENT DE LA TABLE CAR AYANT POUR CAR_ID X, TOUTES LES OPTIONS DE LA TABLE OPTION AYANT POUR CAR_ID X SONT SUPPRIMEES.

3. Avec qui avez-vous travaillé ?

J'ai travailler seul.

4. Contexte

Nom de l'entreprise, organisme ou association ▸

Chantier, atelier, service ▸

Période d'exercice ▸ Du : 01/05 au : 02/05

5. Informations complémentaires *(facultatif)*

Titres, diplômes, CQP, attestations de formation

(facultatif)

Intitulé	Autorité ou organisme	Date

Déclaration sur l'honneur

Je soussigné(e) René Leliard ,

déclare sur l'honneur que les renseignements fournis dans ce dossier sont exacts et que je suis l'auteur(e) des réalisations jointes.

Fait à Saint Victor la Coste le 18/09/2024

pour faire valoir ce que de droit.

Signature :

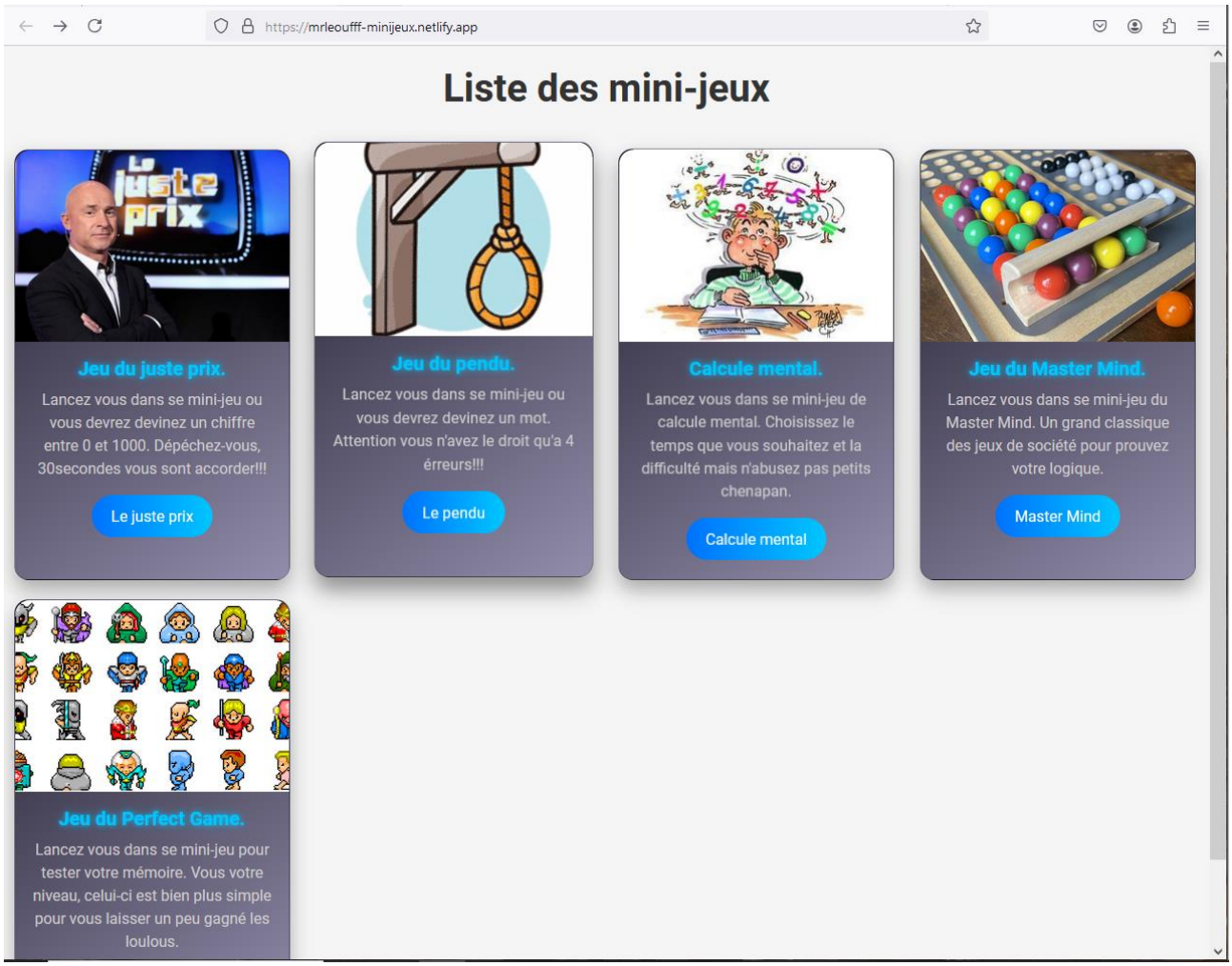
Documents illustrant la pratique professionnelle

(facultatif)

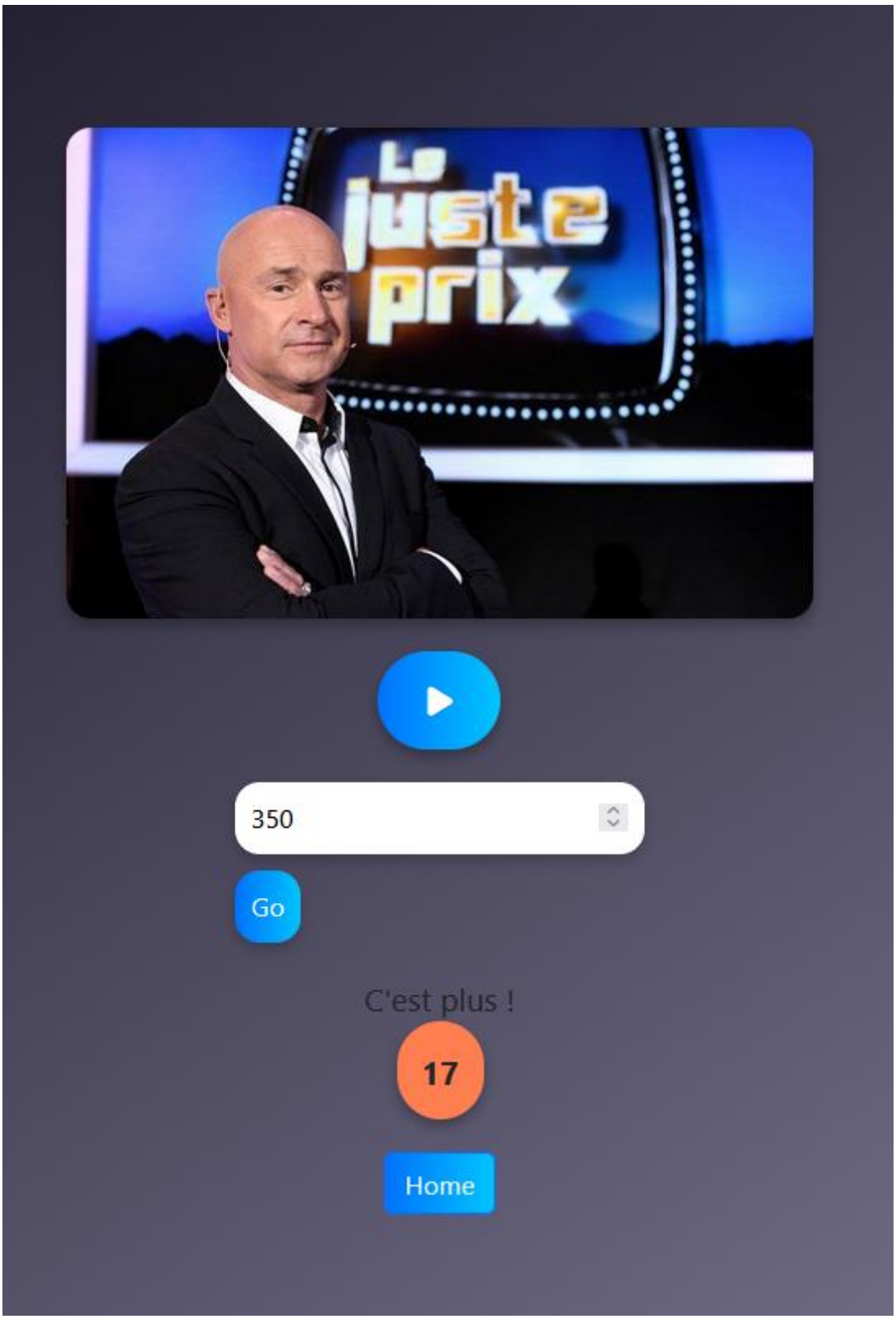
Intitulé

ANNEXES

(Si le RC le prévoit)

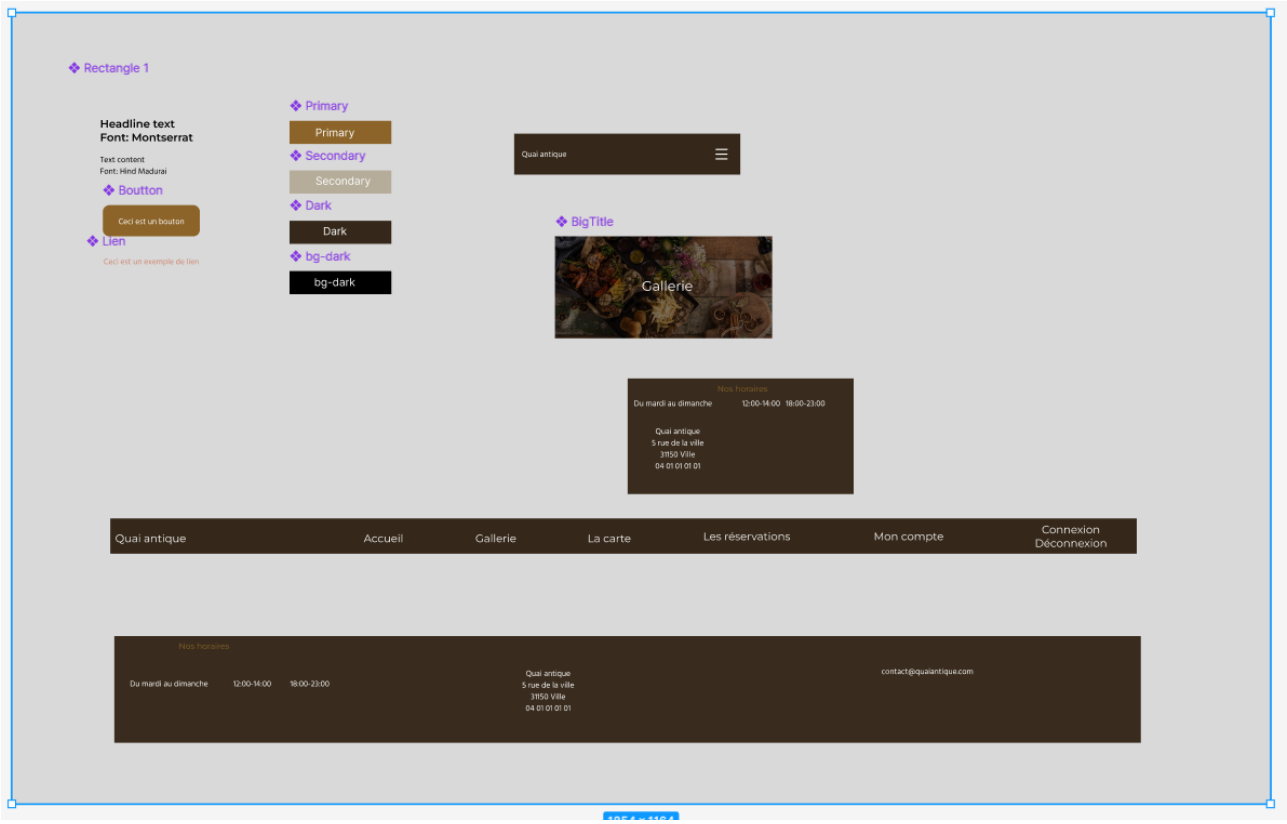



Liste des mini-jeux
Le juste prix
Jeu du juste prix.
Lancez vous dans se mini-jeu ou vous devrez devinez un chiffre entre 0 et 1000. Dépéchez-vous, 30secondes vous sont accorder!!!
Le juste prix


Le juste prix
350
Go
C'est plus !
17
Home

Quai Antique :

Chartre Graphique :



Maquettes :


Mobile
Accueil
Quai antique
Le chef Arnaud Michand vous invite au
Quai Antique
Réserver
Bienvenue au Quai Antique
Bienvenue sur le site de notre restaurant gastronomique situé au coeur de la magnifique ville de chambéry. Le chef Arnaud Michant vous invite à découvrir un voyage culinaire mémorable.
Nos menus
Des produits frais
Découvrez la richesse des saveurs avec nos plats élaborés à partir de produits frais et de saison, cueillis localement.
Voir la carte
Réservation en ligne
Vous pouvez directement réserver votre table sur ce site ! Il suffit de cliquer sur le bouton ci-dessous
Réserver
Nos horaires
Du mardi au dimanche
12:00-14:00 18:00-23:00
Quai antique
5 rue de la ville
31150 Ville
04 01 01 01 01
contact@quaiantique.com
Gallerie
Quai antique
Gallerie
Ajoutez une photo
Titre
Réservation
Nos horaires
Du mardi au dimanche
12:00-14:00 18:00-23:00
Quai antique
5 rue de la ville
31150 Ville
04 01 01 01 01
contact@quaiantique.com


Accueil
Quel antique
Accueil
Galerie
La carte
Les réservations
Mon compte
Connexion
Déconnexion


Gallerie
Accueil
Gallerie
La carte
Mon compte
Connexion
Déconnexion
Gallerie

```sql
CREATE DATABASE IF NOT EXISTS testfacetwit;

USE testfacetwit;

CREATE TABLE users(
    Id_User INT AUTO_INCREMENT ,
    username VARCHAR(50) NOT NULL,
    email VARCHAR(255) NOT NULL UNIQUE,
    role JSON NOT NULL,
    password VARCHAR(255) NOT NULL,
    PRIMARY KEY(Id_User),
    UNIQUE(username)
) ENGINE=InnoDB;

CREATE TABLE post(
    Id_Post INT AUTO_INCREMENT,
    title VARCHAR(50) NOT NULL,
    content TEXT NOT NULL,
    image VARCHAR(255) NOT NULL,
    alt VARCHAR(255) NOT NULL,
    publishedAt DATETIME DEFAULT CURRENT_TIMESTAMP,
    Id_User INT NOT NULL,
    PRIMARY KEY(Id_Post),
    FOREIGN KEY(Id_User) REFERENCES Users(Id_User) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB;

CREATE TABLE post_lang (
    Id_Post_Lang INT AUTO_INCREMENT,
    lang_code CHAR(5) NOT NULL,
    translate_title VARCHAR(50) NOT NULL,
    translate_content TEXT NOT NULL,
    Id_Post INT NOT NULL,
    PRIMARY KEY (Id_Post_Lang),
    FOREIGN KEY (Id_Post) REFERENCES Post(Id_Post) ON DELETE CASCADE ON UPDATE CASCADE,
    UNIQUE (Id_Post, lang_code)
) ENGINE=InnoDB;

```