



MINISTÈRE CHARGÉ  
DE L'EMPLOI

# DOSSIER PROFESSIONNEL (DP)

<i>Nom de naissance</i>	- MORIN
<i>Nom d'usage</i>	- MORIN
<i>Prénom</i>	- PIERRE
<i>Adresse</i>	- 1 PLACE DE LA PYROTECHNIE 18000 BOURGES

## Titre professionnel visé

*Développeur Web et Web Mobile*

### MODALITÉ D'ACCÈS :

- 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>



# DOSSIER PROFESSIONNEL (DP)

## Sommaire

### Exemples de pratique professionnelle

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

p. 5

- CP 1 Installer et configurer son environnement de travail en fonction du projet web ou web mobile p. 5
- CP 2 Maquetter des interfaces utilisateur web ou web mobile p. 8
- CP 3 Réaliser des interfaces utilisateur statiques web ou web mobile p. 13
- CP 4 Développer la partie dynamique des interfaces utilisateur web ou web mobile p. 19

#### Intitulé de l'activité-type n° 2 : Développer la partie back-end d'une application web ou web mobile sécurisée.

p. 26

- CP 5 Mettre en place une base de données relationnelle p. 26
- CP 6 Développer des composants d'accès aux données SQL et NoSQL p. 32
- CP 7 Développer des composants métier côté serveur p. 41
- CP 8 Documenter le déploiement d'une application dynamique web ou web mobile p. 51

### Titres, diplômes, CQP, attestations de formation (*facultatif*)

p. 58

### Déclaration sur l'honneur

p. 59

### Documents illustrant la pratique professionnelle (*facultatif*)

p. 60

### Annexes (*Si le RC le prévoit*)

p. 61

# **EXEMPLES DE PRATIQUE**

## **PROFESSIONNELLE**



MINISTÈRE CHARGÉ  
DE L'EMPLOI

# DOSSIER PROFESSIONNEL (DP)

## Activité-type 1

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

**CP 1 - *Installer et configurer son environnement de travail en fonction du projet web ou web mobile***

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

Pour faire du développement web et web mobile, nous avons besoin de mettre en place un environnement de travail. L'environnement de travail peut être divisé en deux grandes parties : la partie hardware, qui correspond à notre matériel, et la partie software, qui correspond aux logiciels et outils que nous allons utiliser pour faire du développement web.

Côté hardware, j'ai développé les différents projets que vous verrez dans ce Dossier Professionnel sur une machine virtuelle qui tourne sous le système d'exploitation Linux Ubuntu (64-bit).

Côté software, différents logiciels et/ou plateformes installés sur cette machine virtuelle me permettent de faire du développement web dans des conditions optimales.

- Un **éditeur de code** : Visual Studio Code qui me permet d'écrire et de modifier du code. Ses fonctionnalités de débogage, de mise en valeur de la syntaxe ainsi que ses nombreuses extensions en font un logiciel extrêmement complet pour le développement de projets web et web mobile. J'ai par exemple installé l'extension **Material Icon Theme** pour une meilleure clarté dans mes dossiers et fichiers, ainsi que l'extension **Indent Rainbow** pour avoir une indentation plus claire dans mes fichiers.
- Un **logiciel de versionning** : Git, qui, associé à la plateforme GitHub, me permet d'héberger le code de mes différents projets. Côté sécurité, la mise en place d'une clé SSH (Secure Shell) permet d'avoir une méthode d'authentification sécurisée à ses repositories Github en chiffrant les données qui transitent et en vérifiant l'intégrité des données.
- Un **serveur http** : Apache, qui permet une communication avec le navigateur via le protocole http(s).
- Un **système de gestion de base de données** : MariaDB, qui me permet de créer mes bases de données pour mes différents projets. Je gère ensuite mes bases de données via l'interface graphique Adminer.

- Un **langage de programmation interprété**, PHP, pour produire des pages web de façon dynamique. Il est couplé au serveur Apache pour assurer la communication avec le client web.

À noter, à ce stade, j'ai installé ce qu'on appelle un serveur web LAMP (**L**inux, **A**pache, **M**ySQL, **P**HP) fonctionnel qui va me permettre de développer mes projets dans un environnement local.

- Des **outils de gestion de projet**, comme Trello ou Notion, qui me permettent de lister mes tâches, de visionner facilement l'avancement de mes projets et de pouvoir coopérer facilement avec mes collègues lorsque nous travaillons en petits groupes. Pour le travail en équipe, l'utilisation de l'extension VS Code Liveshare permet de mettre en place le pair-programming de façon simple et efficace.
- Une **plateforme de conteneurisation** : Docker pour créer, packager, livrer et exécuter des applications de manière simple, sous forme de conteneurs légers, portables et autonomes, qui peuvent fonctionner pratiquement n'importe où.

## 2. Précisez les moyens utilisés :

La machine virtuelle est installée sur un hôte : mon ordinateur fixe qui tourne sous le système d'exploitation Windows 10.

Les différents logiciels énumérés sont installés via des commandes Linux dans le terminal, permettant de télécharger les softwares sur notre machine virtuelle.

Exemple pour installer le serveur web fonctionnel, j'ai utilisé les commandes suivantes :

- Paquets pour Apache, MariaDB et PHP

```
sudo apt install apache2 php libapache2-mod-php mariadb-server php-mysql
```

- Paquets pour installer les modules les plus courants de PHP

```
sudo apt install php-curl php-gd php-intl php-json php-mbstring php-xml php-zip
```

## 3. Avec qui avez-vous travaillé ?

Pour la mise en place de mon environnement de développement, j'ai travaillé seul.

## 4. Contexte

Nom de l'entreprise, organisme ou association ➤ **Ecole O'clock**

Chantier, atelier, service ➤ **Projet réalisé au cours de la formation O'clock.**

Période d'exercice ➤ Du : **30/10/2023** au : **22/04/2024**



MINISTÈRE CHARGÉ  
DE L'EMPLOI

# DOSSIER PROFESSIONNEL (DP)

## 5. Informations complémentaires (facultatif)

## Activité-type 1

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

CP 2 - *Maquetter des interfaces utilisateur web ou web mobile*

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

Pour cette compétence professionnelle, je vais m'appuyer sur un projet personnel réalisé en parallèle de ma formation chez O'clock. Il y a quelques semaines, la star de l'univers Pokemon, j'ai nommé Pikachu, m'a envoyé une demande un peu spécifique : réaliser un site CV à son effigie. Ayant eu la chance de découvrir la licence de Game Freak dès mes 10 ans, j'ai tout de suite accepté la demande du héros de mon enfance.

J'ai pris ensuite le temps d'échanger avec lui sur les détails du projet pour analyser les besoins et réaliser un site internet sur mesure.

Après quelques échanges, voici les besoins qui sont ressortis de nos discussions :

- Le site devra comporter une page d'accueil, une page présentant les jeux dont il est le héros, une page présentant les différentes évolutions du Pokémons et une page "galerie de photos".
- Pour le moment, la version livrée devra être fournie avec de faux textes en PokemIpsum, c'est Pikachu lui-même qui rédigera les textes du site par la suite.
- Côté graphisme, le site reprendra la couleur fétiche de notre héros : le jaune.
- Enfin, le site devra posséder une version mobile et une version desktop.

### Les user stories, pour ne rien oublier

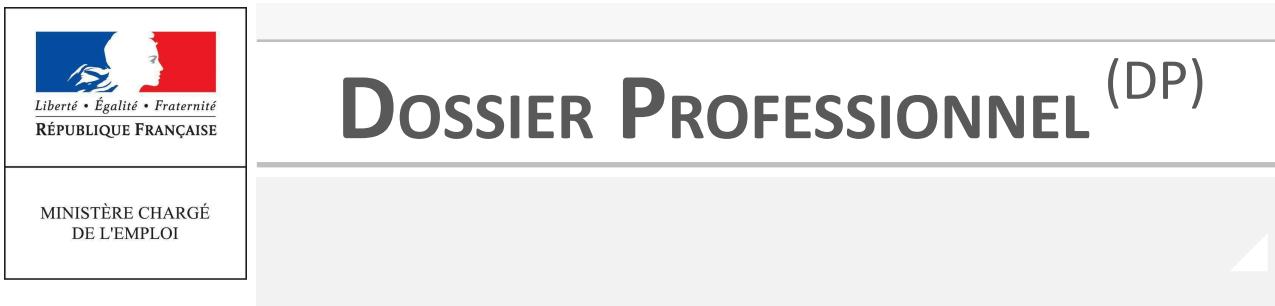
Une fois les bases du projet posées, il convient de rédiger les user stories du site que nous voulons créer. Un récit utilisateur, ou user story, est une phrase simple qui permet de décrire une fonctionnalité du site web que l'on veut créer et rédigée du point de vue de l'utilisateur final. Par exemple : "**En tant qu'utilisateur, je veux pouvoir consulter la page "Jeux" afin de** connaître tous les jeux dont Pikachu est le héros." En fin de projet, cela permet de s'assurer que toutes les fonctionnalités requises ont bien été développées.

Voici les user stories pour le site de Pikachu :

**"En tant qu'utilisateur, je veux pouvoir consulter une page d'accueil avec les dernières infos concernant Pikachu afin de pouvoir connaître les dernières nouvelles concernant mon Pokémons préféré."**

**"En tant qu'utilisateur, je veux pouvoir consulter la page "Jeux" afin de connaître tous les jeux dont Pikachu est le héros."**

**"En tant qu'utilisateur, je veux pouvoir consulter la page "Évolutions" afin de connaître toutes les évolutions de Pikachu."**



# DOSSIER PROFESSIONNEL (DP)

**“En tant qu’utilisateur, je veux pouvoir consulter la page “Photos” afin d’accéder à une galerie de photos de mon Pokémon préféré.”**

**“En tant qu’utilisateur, je veux pouvoir passer d’une page à l’autre grâce à une navbar afin de naviguer de manière fluide sur le site.”**

## Place aux wireframes, pour se projeter

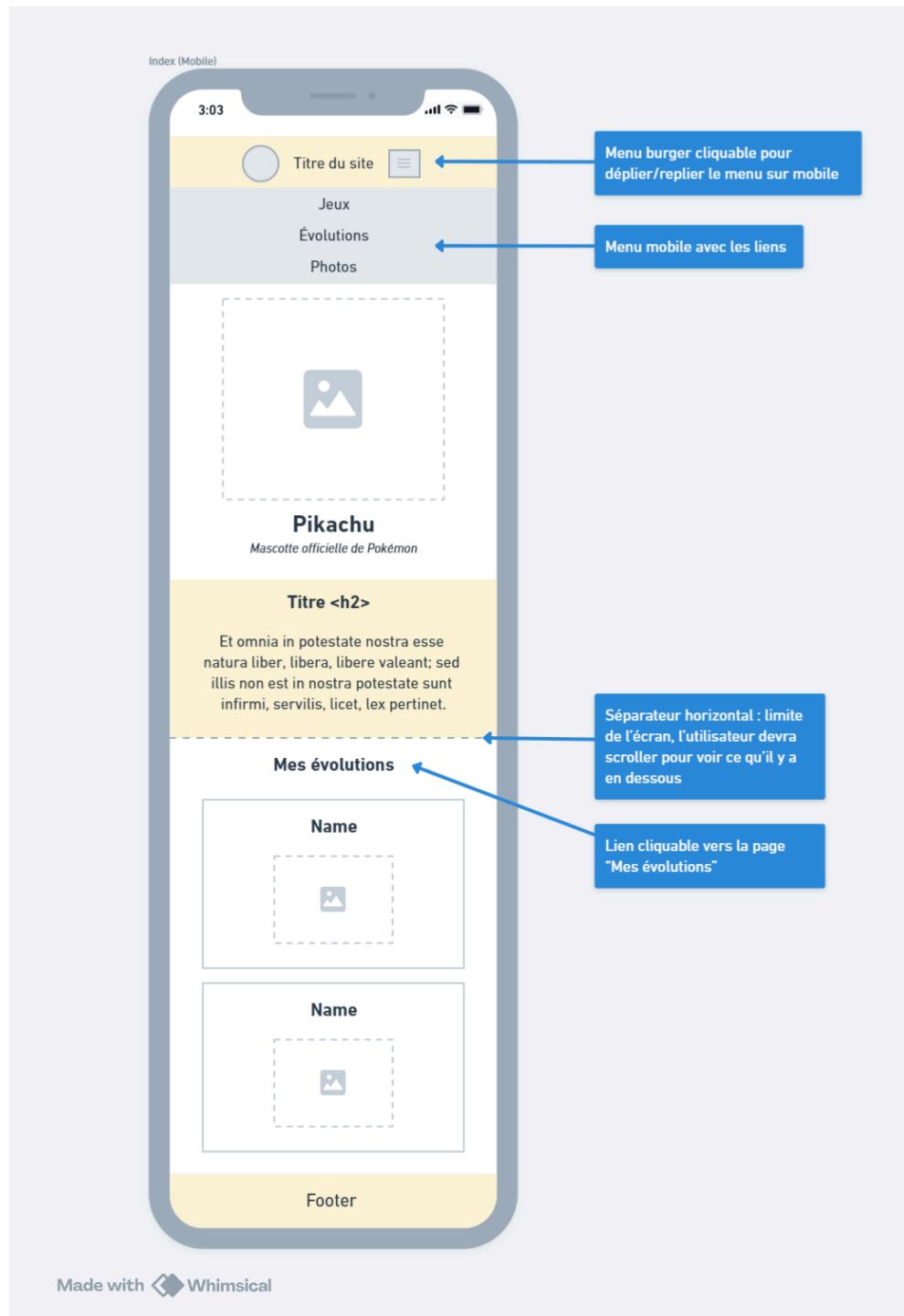
Les user stories sont définies et m’ont permis de bien cerner les contours du projet, je souhaite donc maintenant passer à l’étape suivante et proposer un aperçu graphique du futur site grâce à une série de wireframes. Les wireframes, à ne pas confondre avec les maquettes, sont des schémas utilisés lors de la conception d’une interface utilisateur pour définir les zones et les composants que cette interface doit contenir. À contrario, la maquette est une version aboutie du site que le développeur front-end devra reproduire à l’identique.

Pour ce projet, j’ai réalisé deux wireframes pour chaque page : un wireframe pour la version mobile et un wireframe pour la version desktop.

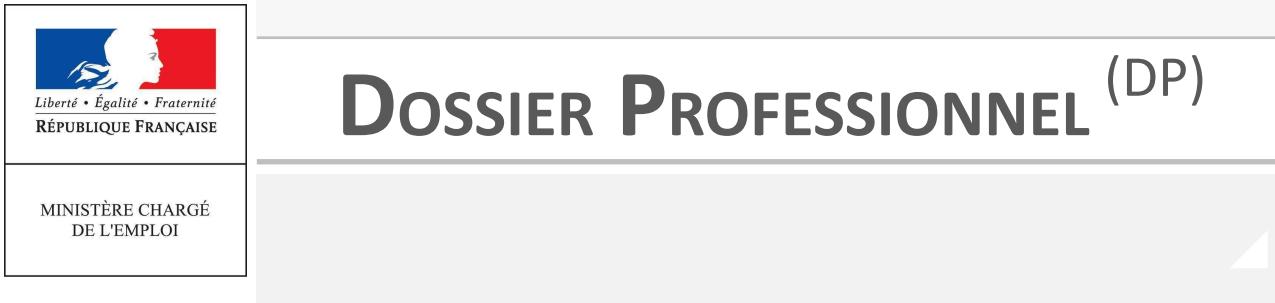
## Wireframes de la page d'accueil du site (vue globale) :

Sur ces wireframes, on peut voir la déclinaison mobile et la déclinaison desktop de la page d'accueil du futur site web de Pikachu.

## Wireframes de la page d'accueil du site (détails version mobile) :



Quand on crée des wireframes, il faut être le plus visuel possible pour que notre client / product owner puisse se projeter. Cela passe donc par la mise en place de formes simples et parlantes. Si nous avons besoin de préciser une fonctionnalité, il convient de placer des annotations comme je l'ai fait ici avec le menu burger ou avec le lien cliquable.



Wireframes de la page d'accueil du site (détails version desktop) :

This detailed wireframe shows the layout of the homepage content area:

- Header:** "Jeux" (highlighted), "Évolutions", "Photos". A blue callout points to the "Navbar avec les liens des différentes pages du site".
- Mascotte officielle de Pokémons:** A placeholder box with a photo icon and the text "Pikachu" below it.
- Text blocks:** Two sections with placeholder text: "Titre <h2>" and "Titre <h2>".
- Section "Mes évolutions":** Contains three cards, each with a placeholder photo icon and the word "Name". A blue callout points to "Lien cliquable vers la page 'Mes évolutions'".
- Footer:** A yellow bar at the bottom.

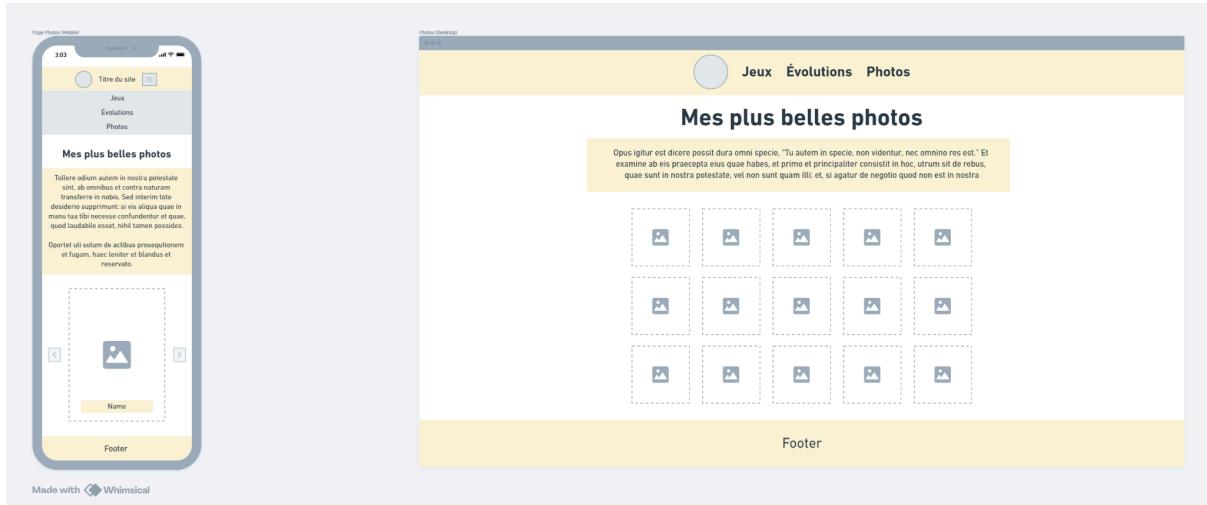
Made with Whimsical

Wireframes des pages jeux, évolutions & photos (vue globale) :

This global view wireframe shows the layout of three main sections:

- Games:** A list of four placeholder game cards, each with a photo icon and the word "Game".
- Evolutions:** A list of four placeholder evolution cards, each with a photo icon and the word "Pokémon".
- Photos:** A list of four placeholder photo cards, each with a photo icon and the word "Photo".

Made with Whimsical



## 2. Précisez les moyens utilisés :

Pour réaliser ces wireframes, j'ai utilisé le site [whimsical.com](https://whimsical.com) qui permet de réaliser facilement des wireframes grâce à la mise à disposition de bibliothèques d'éléments et d'icônes.

## 3. Avec qui avez-vous travaillé ?

Pour ce projet, réalisé en parallèle de mon cursus chez O'clock, j'ai travaillé seul.

## 4. Contexte

Nom de l'entreprise, organisme ou association ➔ **Ecole O'clock**

Chantier, atelier, service ➔ **Projet personnel réalisé en parallèle de mon cursus chez O'clock.**

Période d'exercice ➔ Du : **30/10/2023** au : **22/04/2024**

## 5. Informations complémentaires (facultatif)

Le repository de ce projet personnel est accessible ici :

<https://github.com/PierreMorin4590/cv-pikachu-bootstrap>

Le mini-site web sur Pikachu est accessible ici :

<https://pierremorin4590.github.io/cv-pikachu/>



MINISTÈRE CHARGÉ  
DE L'EMPLOI

# DOSSIER PROFESSIONNEL (DP)

## Activité-type 1

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

CP 3 - *Réaliser des interfaces utilisateur statiques web ou web mobile*

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

Pour cette compétence professionnelle, continuons avec le projet personnel débuté dans la CP 2. Grâce aux users stories et aux wireframes, j'ai pu coder le site statique et mettre en pratique mes connaissances acquises dans les langages HTML (HyperText Markup Language) et CSS (Cascading Style Sheets).

J'ai réalisé ce site en HTML en respectant la sémantique et les bonnes pratiques d'organisation du contenu d'une page web et effectué la mise en page à l'aide du framework CSS Bootstrap.

Pour rendre mon code le plus lisible possible, notamment pour le référencement naturel / SEO (Search Engine Optimization) et pour les lecteurs d'écran des personnes en situation de handicap, je l'ai découpé en blocs logiques qui permettent de bien comprendre la structuration de mon code.

Cette approche permet également à un autre développeur de pouvoir reprendre mon code et de le comprendre rapidement.

Lors de la création d'un site ou d'une application web, il faut penser aux différents moyens d'y accéder : tout le monde ne verra pas le site sur un ordinateur de bureau. Aujourd'hui, avec les smartphones qui sont au cœur de notre quotidien, il y a même de fortes chances pour que la majorité des connexions au site se fassent via mobile.

Il convient donc de développer plusieurs types d'affichage et d'interactions en fonction du device de l'utilisateur final : c'est ce qu'on appelle ***la conception responsive***.

C'est ce que j'avais d'ailleurs prévu en amont lors de la création de nos wireframes.

Prenons par exemple, le menu du site :

**Pikachu**  
Mascotte officielle de Pokémon

**Glitch City**

Pokem ipsum dolor sit amet Mewtwo Strikes Back  
Dewgong Mismagius Gasty Celadon City Grotle.  
Sapphire Abomasnow Harden Electrike Sonic Boom  
Gengar Croconaw. Bulbasaur Noctowl Golduck  
searching far and wide Rufflet Pokemon Pidove.  
Velit esse cillum pariatur Cacturne Hoothoot  
Forretress Lombre I like shorts Togepi.

**Pikachu**  
Mascotte officielle de Pokémon

**Glitch City**

Pokem ipsum dolor sit amet Mewtwo Strikes Back  
Dewgong Mismagius Gasty Celadon City Grotle.  
Sapphire Abomasnow Harden Electrike Sonic Boom  
Gengar Croconaw. Bulbasaur Noctowl Golduck

Sur la version mobile ci-dessus, on voit que le menu de navigation du site est caché (capture 1) et qu'il n'apparaît que lorsque je clique sur le menu burger (capture 2).

Sur la version desktop ci-dessous, le menu de navigation est visible par défaut car la résolution permet un affichage des composants sur un espace beaucoup plus large.

**Pikachu**  
Mascotte officielle de Pokémon

**Glitch City**

Fog Badge Aggron Ash Ketchum Joltik Snubbull the power that's inside Bastiodon. Scratch Elektross Scizor MysteryBerry Pineco Lopunny Blissey. Glitch City Masquerain Nincada like no one ever was Doduo Quagsire Whimsicott. Sed do eiusmod tempor incididunt Uxie Poison Sting Magby Durant Linone Lotad. Celadon City Camerupt Sigilyph Rampardos Vermilion City Oillery Youngster wants to fight.

**Inside Bastiodon**

The Power Of One Lampent Blaziken Herdier Electrike Pansage Salamence. Gold Ground Mint Berry Woobat Johto Sharpedo Metang. Water Gun Chimchar Heatmor Cryogonal Electrode Primeape Joltik. Leech Seed Wallmer Escape Rope Machoke Magmarite Deino Arbok. Psychic to extend our reach to the stars above Minccino Stoutland Gothita Poison Blaziken.

**Sapphire Luvdisc**

Rainbow Badge we're blasting off again Brock Shaymin Jesse Kadabra Victini. Sapphire Luvdisc Corphish Whiscash Snorlax Wartortle Slowpoke. Duis aute irure dolor in reprehenderit in voluptate Tropius Moltres Tirtouga Skorupi Reunicus Audino. Grass Zapdos Ground Pineco Magikarp Mareep Spoink. Poison Dewgong Gohgorita Geodude Pawniard Beartic Lairon.

**Mes évolutions**

**Pichu**   **Raichu**   **Raichu d'Alola**



MINISTÈRE CHARGÉ  
DE L'EMPLOI

# DOSSIER PROFESSIONNEL (DP)

## Un code clair et organisé

Comme expliqué précédemment, le code HTML est découpé en bloc “logique” pour respecter la sémantique du langage HTML. Prenons le bloc <body> de la page d'accueil:

```
<body>
  <div class="wrapper">
    <header> ...
    </header>
    <main class="content"> ...
    </main>
    <footer class="footer bg-warning"> ...
    </footer>
  </div>
</body>
```

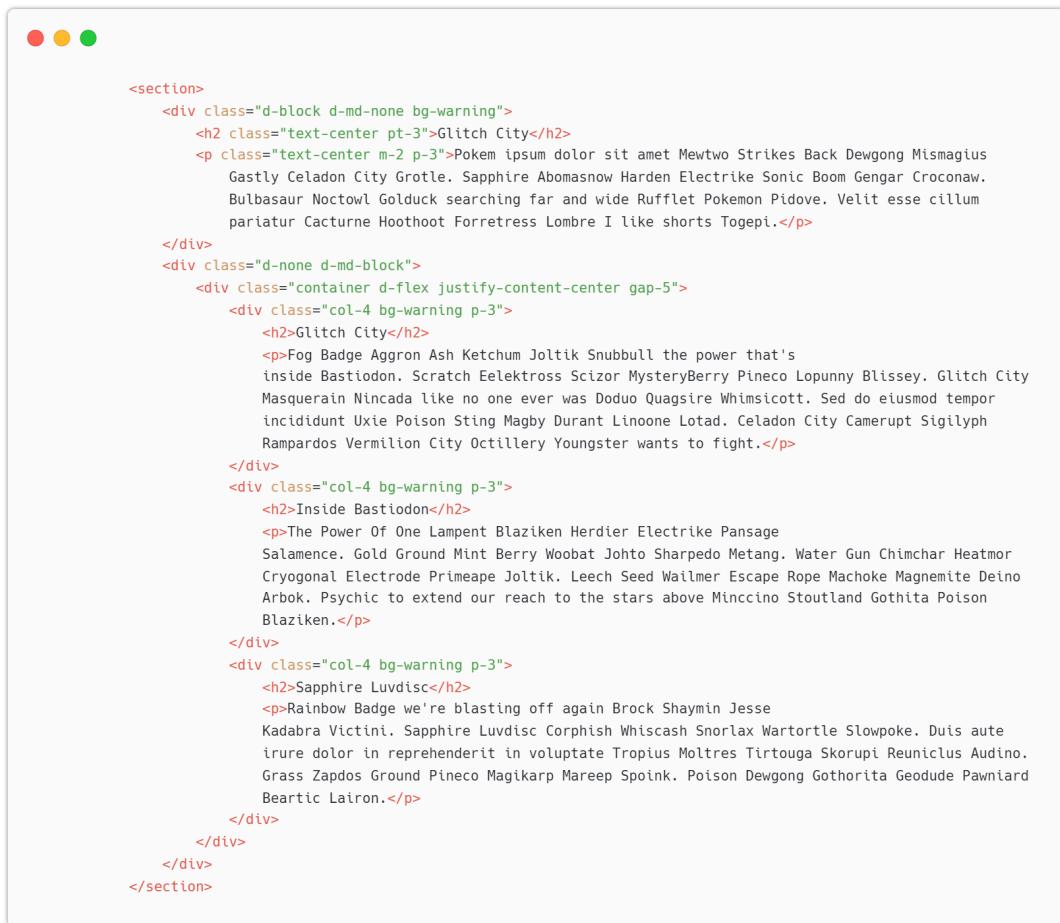
On voit ici que le bloc <body> est découpé en 3 sous-blocs logiques : un <header> qui va permettre notamment de placer une balise <h1>, un <main> où sera développé le contenu de la page et un <footer> pour les informations de pied-de-page.

Faire du HTML sémantique est très important pour de nombreuses raisons comme le référencement ou l'accessibilité par exemple. Et mon code sera aussi facilement compréhensible par les futurs développeurs qui arriveront sur le projet.

## Le responsive, comment ça marche ?

Pour que le site soit responsive, c'est à dire qu'il s'adapte au format du device utilisé (mobile ou desktop, en l'occurrence), il faut mettre en place des *media queries* : un outil de responsive design qui permet d'adapter la feuille de style CSS en fonction de différents paramètres ou caractéristiques de l'appareil. On peut, par exemple, appliquer différentes mises en forme selon la taille de la zone d'affichage de l'appareil utilisé.

Prenons un exemple sur notre page d'accueil :



```
<section>
  <div class="d-block d-md-none bg-warning">
    <h2 class="text-center pt-3">Glitch City</h2>
    <p class="text-center m-2 p-3">Pokem ipsum dolor sit amet Mewtwo Strikes Back Dewgong Mismagius
      Gasty Celadon City Grotle. Sapphire Abomasnow Harden Electrike Sonic Boom Gengar Croconaw.
      Bulbasaur Noctowl Golduck searching far and wide Rufflet Pokemon Pidove. Velit esse cillum
      pariatur Cacturne Hoothoot Forretress Lombre I like shorts Togepi.</p>
  </div>
  <div class="d-none d-md-block">
    <div class="container d-flex justify-content-center gap-5">
      <div class="col-4 bg-warning p-3">
        <h2>Glitch City</h2>
        <p>Fog Badge Aggron Ash Ketchum Joltik Snubbull the power that's
          inside Bastiodon. Scratch Elektrross Scizor MysteryBerry Pineco Lopunny Blissey. Glitch City
          Masquerain Nincada like no one ever was Doduo Quagsire Whimsicott. Sed do eiusmod tempor
          incididunt Uxie Poison Sting Magby Durant Linoone Lotad. Celadon City Camerupt Sigilyph
          Rampards Vermilion City Octillery Youngster wants to fight.</p>
      </div>
      <div class="col-4 bg-warning p-3">
        <h2>Inside Bastiodon</h2>
        <p>The Power Of One Lampent Blaziken Herdier Electrike Pansage
          Salamence. Gold Ground Mint Berry Woobat Johto Sharpedo Metang. Water Gun Chimchar Heatmor
          Cryogonal Electrode Primeape Joltik. Leech Seed Wailmer Escape Rope Machoke Magnemite Deino
          Arbok. Psychic to extend our reach to the stars above Minccino Stoutland Gothita Poison
          Blaziken.</p>
      </div>
      <div class="col-4 bg-warning p-3">
        <h2>Sapphire Luvdisc</h2>
        <p>Rainbow Badge we're blasting off again Brock Shaymin Jesse
          Kadabra Victini. Sapphire Luvdisc Corphish Whiscash Snorlax Wartortle Slowpoke. Duis aute
          irure dolor in reprehenderit in voluptate Tropius Moltres Tirtouga Skorupi Reuniclus Audino.
          Grass Zapdos Ground Pineco Magikarp Mareep Spink. Poison Dewgong Gothorita Geodude Pawniard
          Beartic Lairon.</p>
      </div>
    </div>
  </div>
</section>
```

Dans cette section, nous avons un affichage conditionnel selon la largeur du device de l'utilisateur. Pour cela, j'ai utilisé les class [display property](#) du framework Bootstrap qui permettent de jouer avec les breakpoints. J'ai utilisé le breakpoint **md** de Bootstrap qui correspond aux écrans supérieurs à **768 pixels** de large.

Observons et expliquons les class de la div : `<div class="d-block d-md-none bg-warning">`

- La class d-block indique que la div sera visible sur la page.
- La class d-md-none indique qu'au breakpoint md, la div ne sera plus visible.

De même pour la div : `<div class="d-none d-md-block">`

- La class d-none indique que la div ne sera pas visible sur la page.
- La class d-md-block indique qu'au breakpoint md, la div sera plus visible.

Ainsi, sur des écrans mobiles, j'affiche un résumé (un paragraphe) du contenu qui est visible sur la version desktop (3 blocs de texte).



# DOSSIER PROFESSIONNEL (DP)

MINISTÈRE CHARGÉ  
DE L'EMPLOI

À titre d'exemple, voici ce qu'il faudrait faire en CSS Vanilla pour obtenir le même résultat que Bootstrap pour cette div <div class="d-none d-md-block"> :

```
● ● ●  
  
@media screen and (max-width: 768px) {  
    div {  
        display: none;  
    }  
}
```

Ici, les styles à l'intérieur de la règle de Media Query ne seront appliqués que lorsque la largeur de l'écran est de 768 pixels ou moins. Cela correspond donc bien aux class Bootstrap **d-none** & **d-md-block** que j'ai appliquées à ma <div>.

## 2. Précisez les moyens utilisés :

Pour réaliser ce site, j'ai utilisé l'éditeur de code Visual Studio Code.

Pour tester l'affichage de la page web, j'ai utilisé le navigateur web Google Chrome et notamment ses outils d'aide au développement pour faciliter l'intégration et le débogage de mon code.

J'ai également utilisé la documentation en ligne de la MDN afin d'être le plus précis possible dans la sémantique des balises HTML mais aussi pour connaître et comprendre le fonctionnement des propriétés CSS.

- <https://developer.mozilla.org/fr/docs/Web/HTML>
- <https://developer.mozilla.org/fr/docs/Web/CSS>

Pour la mise en forme CSS, j'ai utilisé le framework CSS Bootstrap (version 5.3) et sa documentation :

- <https://getbootstrap.com/docs/5.3/getting-started/introduction/>

## 3. Avec qui avez-vous travaillé ?

Pour ce projet, réalisé en parallèle de mon cursus chez O'clock, j'ai travaillé seul.

#### 4. Contexte

Nom de l'entreprise, organisme ou association ➤ **Ecole O'clock.**

Chantier, atelier, service ➤ **Projet personnel réalisé en parallèle de mon cursus chez O'clock.**

Période d'exercice ➤ Du : **30/10/2023** au : **22/04/2024**

#### 5. Informations complémentaires (facultatif)

Le repository de ce projet personnel est accessible ici :

<https://github.com/PierreMorin4590/cv-pikachu-bootstrap>

Le mini-site web sur Pikachu est accessible ici :

<https://pierremorin4590.github.io/cv-pikachu/>



MINISTÈRE CHARGÉ  
DE L'EMPLOI

# DOSSIER PROFESSIONNEL (DP)

## Activité-type 1

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

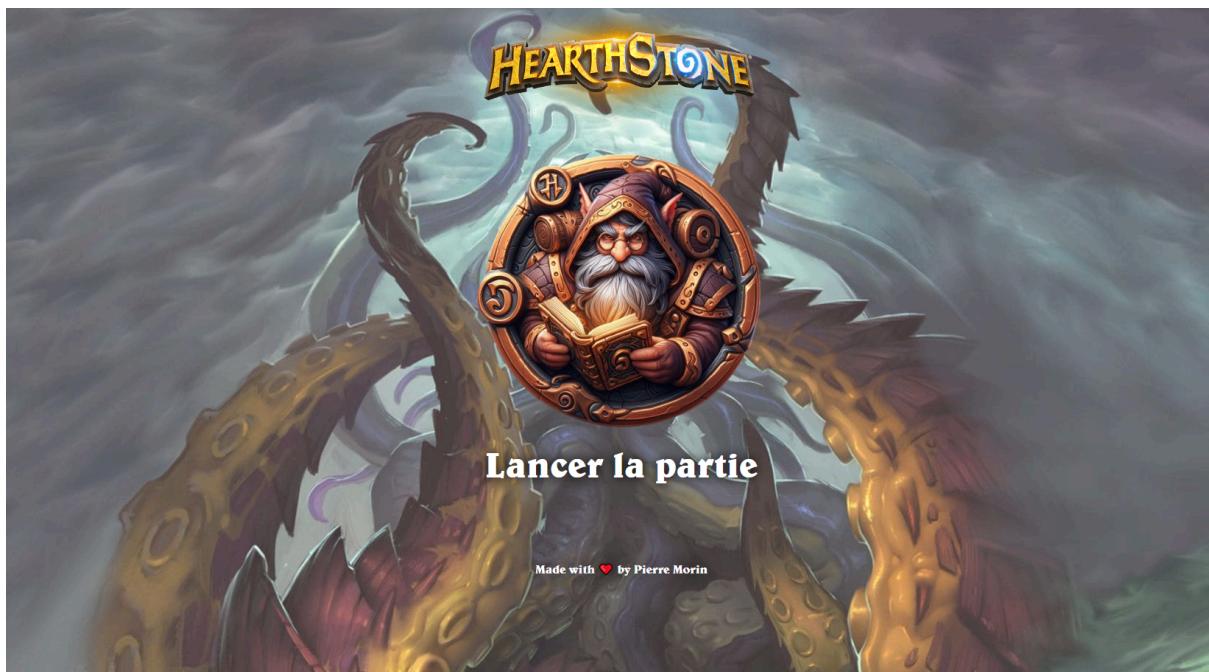
**CP 4 - Développer la partie dynamique des interfaces utilisateur web ou web mobile**

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

En parallèle de la formation de développeur web et web mobile chez O'clock, j'ai réalisé quelques projets personnels comme le jeu de memory inspiré de l'univers du jeu vidéo Hearthstone.

Ce projet a été réalisé en JavaScript Vanilla et permet d'illustrer la dynamisation d'une interface utilisateur. Le jeu créé est un jeu de memory classique où l'on doit retrouver des paires de cartes, cependant chaque partie est unique puisque les cartes sont sélectionnées aléatoirement parmi toutes les cartes disponibles dans l'API [HearthstoneJSON.com](https://HearthstoneJSON.com) grâce à l'API Fetch.

#### Quelques précisions sur les fonctionnalités du jeu



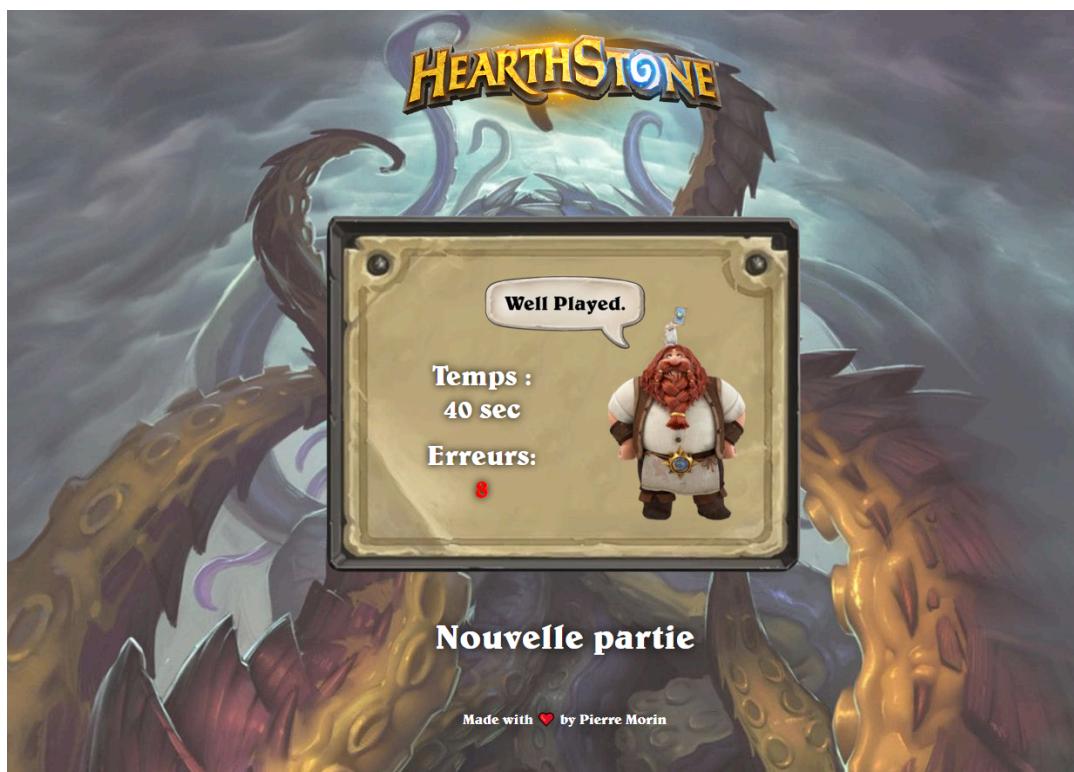
Quand on arrive sur le site, la page d'accueil nous plonge dans l'univers d'Hearthstone et nous invite à lancer une partie.

Voici ce qui se passe quand on lance une partie :



Une série de cartes sélectionnées aléatoirement apparaît et reste visible pendant 5 secondes et les cartes sont ensuite retournées face cachée. Le joueur peut alors commencer à chercher les paires de cartes.

- Si deux cartes retournées sont similaires, elles restent découvertes.
- Si deux cartes retournées ne sont pas similaires, elles sont retournées face cachée.



Une fois la dernière paire retrouvée, le jeu disparaît et une fenêtre de score s'ouvre en indiquant au joueur le temps qu'il a mis à réussir la partie et le nombre d'erreurs commises. À partir de cette fenêtre, le joueur peut retourner sur l'écran d'accueil afin de relancer une nouvelle partie.



# DOSSIER PROFESSIONNEL (DP)

MINISTÈRE CHARGÉ  
DE L'EMPLOI

## Et du côté du code JavaScript, comment ça se passe ?

Pour illustrer le développement de la partie dynamique des interfaces utilisateurs web, je vais expliquer quelques fonctionnalités du jeu Hearthstone Memory présenté précédemment.

- Récupérer les données à l'aide de l'API Fetch :

Quand on clique sur “Lancer la partie”, une série de cartes est sélectionnée aléatoirement et montrée face visible pendant 5 secondes. Pour sélectionner une série de cartes de façon aléatoire, il nous faut dans un premier temps récupérer les données de l'API [HearthstoneJSON.com](https://api.hearthstonejson.com/v1/190920/frFR/cards.collectible.json).

Voici le code qui me permet de récupérer les données au format JSON :

```
const app = {

    // URL de l'API permettant de récupérer les datas Hearthstone
    apiUrl: "https://api.hearthstonejson.com/v1/190920/frFR/cards.collectible.json",
    data: null,

    init: async function () {
        console.log('app.init()');

        this.data = await this.getData();
        console.log(this.data);

        const playButton = document.querySelector(".playButton");
        playButton.addEventListener("click", (event) => this.handlePlayGame(event));
    },

    /**
     * Fonction asynchrone qui récupère les données de l'API HearthstoneJSON api.hearthstonejson.com/v1/
     * @returns {array} apiHearthstone : Tableau avec toutes les cartes Hearthstone ever <3
     */
    getData: async function () {
        try {
            // fetch retourne une promesse (objet)
            const response = await fetch(this.apiUrl);
            if (!response.ok) {
                throw new Error(`Failed to fetch : ${response.status}`);
            }

            // On attend la résolution de la promesse et on récupère les données
            const apiHearthstone = await response.json();

            return apiHearthstone;
        } catch (error) {
            throw new Error(error.message);
        }
    }
}
```

J'utilise un script JavaScript dans lequel une méthode asynchrone `getData` va me permettre de récupérer les données de l'API au format JSON grâce à l'[API Fetch](#). La méthode `fetch()` va permettre d'accéder aux ressources récupérées de manière asynchrone en émettant une requête et en attendant une réponse.

Mon script contient deux propriétés :

- `apiUrl`, qui est l'url d'accès à l'API [HearthstoneJSON.com](#)
- `data`, qui est null par défaut et qui va nous permettre de stocker les données récupérées de l'API

J'ai ensuite une méthode asynchrone `init` qui a pour fonction d'initialiser l'application. Dans cette méthode, j'ai utilisé deux fois la méthode `console.log()` pour vérifier :

1. Dans un premier temps que ma méthode `init` se lance bien avec :  
`console.log('app.init()');`
2. Dans un second temps pour vérifier que j'ai bien récupéré mes données avec :  
`console.log(this.data);`



```
app.init() app.js:16
app.js:19
(5845) [{}]
```

The screenshot shows the output of the `app.init()` function. It consists of a single line of code at the top, followed by a large array of 5845 elements. Each element is represented by a single brace {}, indicating an object. The array starts with '(5845) [{}' and ends with '...]'.

Nous voyons donc ici, grâce à la capture effectuée dans la console des [Outils de développement de Google Chrome](#) que nos actions sont bien effectuées : la méthode `init` se lance bien et nous récupérons bien les données des 5845 cartes Hearthstone de l'API [HearthstoneJSON.com](#).

Enfin, un écouteur d'événement avec la méthode `addEventListener()` a été placé sur la <div> portant la class "`playButton`". La méthode `addEventListener()` suit la syntaxe suivante :

```
element.addEventListener(event(required), function(required), useCapture(optional))
```

Deux paramètres sont obligatoires (`required`) pour cette méthode :

1. **event**, le nom de l'événement que je souhaite écouter : ici, ce sera le `click`  
→ quand on clique sur le bouton, une fonction sera exécutée
2. **function**, la fonction qui doit être exécutée quand mon événement est déclenché  
→ ici ce sera la fonction `handlePlayGame()` comme indiqué dans la méthode `init`



# DOSSIER PROFESSIONNEL (DP)

MINISTÈRE CHARGÉ  
DE L'EMPLOI

- **Lancer une partie de Hearthstone Memory :**

Maintenant que nous avons vu comment récupérer les data de l'API à l'initialisation de l'application, voyons ce qui se passe lorsque la fonction ***handlePlayGame()*** est exécutée au moment où l'on clique sur "Lancer la partie". C'est ici que nous allons voir le dynamisme possible dans les interfaces utilisateurs web.

```
/*
 * Fonction qui permet de lancer une partie
 * @param {*} event
 */
handlePlayGame: function (event) {
    const data = document.querySelector(".data");
    data.classList.remove("hidden");
    const playButton = document.querySelector(".playButton");
    playButton.classList.add("hidden");
    const memory = document.querySelector(".memory");
    memory.classList.remove("hidden");

    // Obtenir de nouvelles cartes aléatoires
    const randomCards = this.selectRandomCards(9);

    // Mélanger les cartes
    const shuffledCards = this.shuffleCards(randomCards);

    this.startGame(shuffledCards);
    this.startTimer();
    console.log(playButton);
}
```

Quand je clique sur "Lancer la partie", la fonction ***handlePlayGame()*** est exécutée et déclenche une série d'actions prévues dans le script. Analysons ces éléments dans l'ordre :

1. Tout d'abord, nous voyons qu'une série de trois actions est effectuée pour modifier le DOM. Le DOM, [Document Object Model](#), est une interface de programmation (ou API, Application Programming Interface, dans la langue de Shakespeare) qui permet à des scripts d'examiner et de modifier le contenu du navigateur web.

Ici, nous voyons très rapidement que les trois actions effectuées ont pour but de retirer ou d'ajouter la class "***hidden***" à trois éléments du DOM. Le but est de faire disparaître l'écran de lancement du jeu et de faire apparaître les cartes du memory.

- La deuxième action de cette fonction est la sélection de cartes aléatoires. Pour cela, la fonction fait appel à une autre fonction de ce script ***selectRandomCards(count)*** qui est une fonction qui a pour but de sélectionner un nombre X de cartes aléatoires parmi les 5845 cartes Hearthstone de l'API [HearthstoneJSON.com](https://HearthstoneJSON.com). La fonction retourne un tableau ***randomCards*** avec les X cartes sélectionnées.

Cela permet de proposer des parties uniques dès qu'on relance le jeu.



```

    /**
     * Fonction qui sélectionne un nombre donné de cartes au hasard parmi les données récupérées
     * @param {number} count : Nombre de cartes à sélectionner
     * @returns {array} randomCards : Tableau des cartes sélectionnées
     */
    selectRandomCards: function (count) {
        const randomCards = [];
        const totalCards = this.data.length;

        for (let i = 0; i < count; i++) {
            const randomIndex = Math.floor(Math.random() * totalCards);
            randomCards.push(this.data[randomIndex]);
        }

        return randomCards;
    }

```

- La troisième action de cette fonction est de mélanger les cartes précédemment sélectionnées. Pour cela, on fait encore appel à une autre fonction de ce script ***shuffleCards(cardSelected)*** et on fait passer le tableau ***randomCards***, que la fonction ***selectRandomCards(9)*** a retourné juste avant, comme paramètre de cette fonction. La fonction retourne un tableau ***cardSet*** avec les 18 cartes mélangées.



```

    /**
     * Fonction qui double puis mélange les cartes
     * @param {array} cardSelected : Tableau qui passe les cartes qu'on veut mélanger
     * @returns {array} cardSet : Tableau avec les cartes mélangées (index random)
     */
    shuffleCards: function (cardSelected) {
        // On double chaque carte
        cardSet = cardSelected.concat(cardSelected);

        for (let i = 0; i < cardSet.length; i++) {
            // On randomise l'index
            let j = Math.floor(Math.random() * cardSet.length);
            // On mélange
            let temp = cardSet[i];
            cardSet[i] = cardSet[j];
            cardSet[j] = temp;
        }

        console.log(cardSet);
        return cardSet;
    }

```

- Enfin les deux dernières actions de cette fonction sont de lancer le jeu avec la fonction ***startGame(shuffledCards)***, qui passe en paramètre le tableau des cartes mélangées ***cardSet*** de la fonction ***shuffleCards(cardSelected)***, et de lancer un timer avec la fonction ***startTimer()*** pour calculer le temps que va mettre le joueur à résoudre la grille de memory.



# DOSSIER PROFESSIONNEL (DP)

MINISTÈRE CHARGÉ  
DE L'EMPLOI

## 2. Précisez les moyens utilisés :

Pour réaliser ce jeu de memory ancré dans l'univers d'Hearthstone, j'ai utilisé l'éditeur de code Visual Studio Code.

Pour tester l'affichage de la page web, j'ai utilisé le navigateur web Google Chrome et notamment ses outils d'aide au développement pour faciliter l'intégration et le débogage de mon code.

J'ai également utilisé la documentation en ligne de la MDN afin d'être le plus précis possible dans la sémantique des balises HTML, pour connaître et comprendre le fonctionnement des propriétés CSS et pour utiliser au mieux le langage de script JavaScript.

- <https://developer.mozilla.org/fr/docs/Web/HTML>
- <https://developer.mozilla.org/fr/docs/Web/CSS>
- <https://developer.mozilla.org/fr/docs/Web/JavaScript>

Toutes les données des cartes Hearthstone viennent de l'API [HearthstoneJSON.com](https://HearthstoneJSON.com).

## 3. Avec qui avez-vous travaillé ?

Pour ce projet, réalisé en parallèle de mon cursus chez O'clock, j'ai travaillé seul.

## 4. Contexte

Nom de l'entreprise, organisme ou association ➤ **Ecole O'clock**

Chantier, atelier, service ➤ **Projet personnel réalisé en parallèle de mon cursus chez O'clock.**

Période d'exercice ➤ Du : **30/10/2023** au : **22/04/2024**

## 5. Informations complémentaires (facultatif)

Le repository de ce projet personnel est accessible ici :

<https://github.com/PierreMorin4590/hearthstone-memory>

Le jeu est accessible ici :

<https://pierremorin4590.github.io/hearthstone-memory/>

## Activité-type 2

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

### CP 5 - Mettre en place une base de données relationnelle

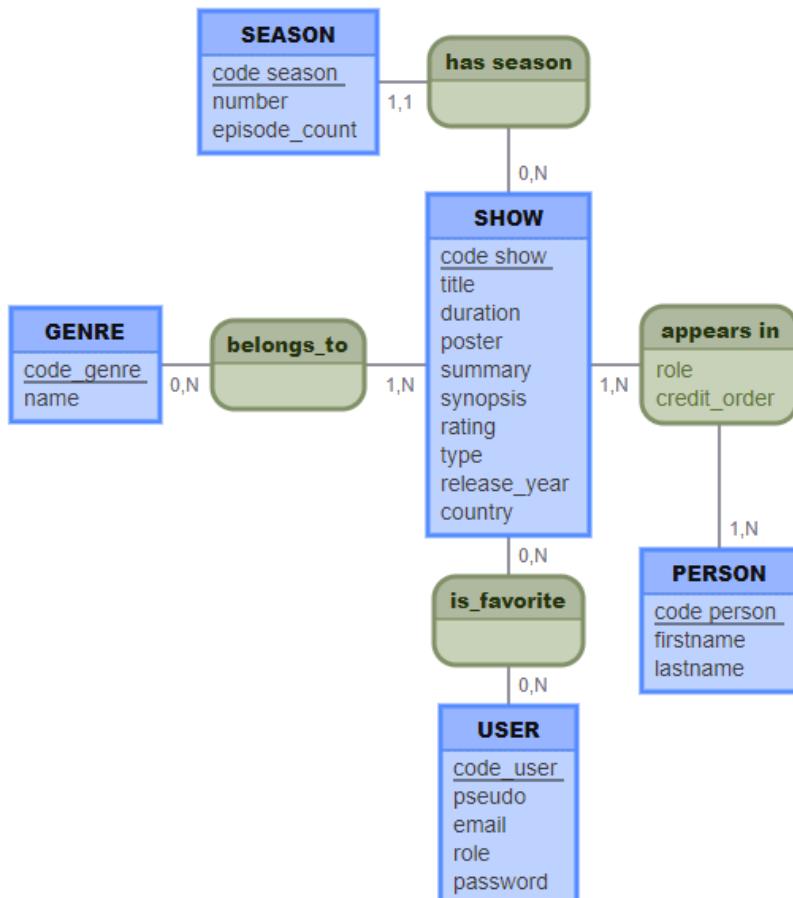
#### 1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

Dans le cadre d'un projet réalisé lors de notre spécialisation sur le framework PHP Symfony à la fin de notre formation nous avons créé O'flix, une base de données de films et séries semblable à un site comme [IMDB](#) :

- les utilisateurs peuvent se connecter et mettre des films/séries en favoris, noter et laisser un commentaire sur un film ou une série,
- les administrateurs ont accès à un backoffice qui permet de créer un nouveau film ou une nouvelle série.

#### Le MCD, Modèle Conceptuel de Données

Pour visualiser au mieux ce projet et comprendre les relations entre les différentes tables de la base de données à créer, j'ai créé un MCD : un Modèle Conceptuel des Données.





# DOSSIER PROFESSIONNEL (DP)

Le modèle conceptuel des données est une représentation des données qui permet de décrire facilement les données utilisées par le système d'information et leurs relations.

Les informations y sont représentées de façon logique grâce à des règles et des diagrammes codifiés :

- Les entités : 1 rectangle correspond à un objet.  
exemple : Le rectangle **PERSON** est une entité.
- Les propriétés : la liste des données liées à une entité.  
exemple : L'entité **PERSON** a trois propriétés → code\_person, firstname & lastname.
- Les relations : comment les entités sont reliées entre elles.  
exemple : Le rectangle vert au bord arrondis **appears in** décrit la relation existante entre les deux entités **SHOW** & **PERSON**.
- Les cardinalités : ce sont des caractères (0, 1, N) qui fonctionnent par couple et qui sont présents de chaque côté d'une relation.  
exemple : Les cardinalités de chaque côté de la relation **appears in** sont 1,N et 1,N. Cela peut se lire de la façon suivante : une **PERSON** peut apparaître dans un ou plusieurs **SHOW** & un **SHOW** peut avoir une ou plusieurs **PERSON**. On remarque d'ailleurs que la relation appears in qui lie ces deux tables est une relation d'association qui possède deux propriétés : role et credit\_order. Ce sont des propriétés qui sont spécifiques à cette relation et qui ne peuvent pas être attribuées à l'une des deux entités.

## Le MLD, Modèle Logique de Données

Une fois le MCD défini, je peux mettre en place un Modèle Logique de Données, ou MLD, qui est la représentation des données d'un système d'information. À partir du MCD, le MLD introduit donc les éléments, les relations et les détails contextuels qui vont être essentiels à la structuration des données. Voici le MLD correspondant à mon MCD :

```
TABLE SHOW (code_show, title, duration, poster, summary, synopsis, rating, type, release_year, country, #code_genre, #code_season)
TABLE SEASON (code_season, number, episode_count, #code_show)
TABLE GENRE (code_genre, name)
TABLE PERSON (code_person, firstname, lastname)
TABLE USER (code_user, pseudo, email, role, password)
```

TABLE **CASTING** (code\_casting, role, credit\_order, #code\_show, #code\_person)

TABLE **FAVORITE** (code\_favorite, #code\_user, #code\_show)

Côté vocabulaire et conventions :

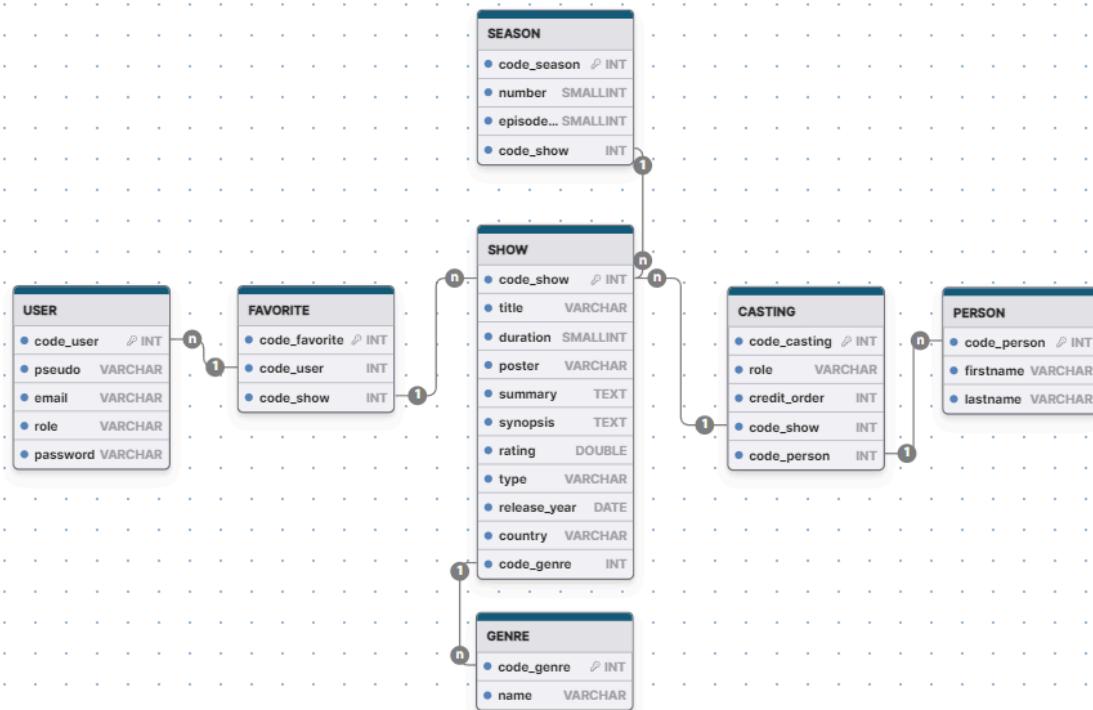
- On note que les entités du MCD deviennent des tables dans le MLD.
- Les clés principales, appelées clés primaires, sont soulignées.
- Les clés étrangères, c'est à dire une clé qui fait référence à la clé primaire d'une autre table, sont représentées en étant préfixées du symbole # dièse.

### Le MPD, Modèle Physique de Données

Après avoir mis par écrit notre MLD, je peux passer à l'étape suivante : la création du Modèle Physique, ou MPD. Le MPD permet de construire la structure finale de la base de données où l'on pourra visualiser les différents liens entre les éléments. Le vocabulaire évolue par rapport au MCD :

- Les entités deviennent des tables.
- Les propriétés deviennent des champs ou des attributs.
- Les propriétés qui se trouvent au milieu d'une relation (comme pour notre relation *appears in* qui possède les propriétés role et credit\_order) sont à l'origine d'une nouvelle table ou vont dans la table appropriée en fonction des cardinalités de la relation.
- Les identifiants deviennent des clés : chaque table doit au minimum avoir une clé primaire.
- Les relations et les cardinalités deviennent des champs ou attributs : elles deviennent des clés étrangères, c'est-à-dire des clés qui font référence à la clé primaire d'une autre table.

Voici mon MPD réalisé grâce à l'app [drawDB](#) :





MINISTÈRE CHARGÉ  
DE L'EMPLOI

# DOSSIER PROFESSIONNEL (DP)

Une fois le MPD réalisé, l'outil que j'ai utilisé me permet d'exporter mon MPD en fichier SQL pour créer directement ma base de données. J'ai pour ma part exporté le fichier dans un format compatible avec MariaDB :

```
CREATE OR REPLACE TABLE `SHOW` (
    `code_show` INT NOT NULL AUTO_INCREMENT UNIQUE,
    `title` VARCHAR(255) NOT NULL,
    `duration` SMALLINT NOT NULL,
    `poster` VARCHAR(255) NOT NULL,
    `summary` TEXT(65535) NOT NULL,
    `synopsis` TEXT(65535) NOT NULL,
    `rating` DOUBLE,
    `type` VARCHAR(10) NOT NULL,
    `release_year` DATE,
    `country` VARCHAR(100) NOT NULL,
    `code_genre` INT NOT NULL,
    PRIMARY KEY(`code_show`)
);
CREATE OR REPLACE TABLE `GENRE` (
    `code_genre` INT NOT NULL AUTO_INCREMENT UNIQUE,
    `name` VARCHAR(50) NOT NULL,
    PRIMARY KEY(`code_genre`)
);
CREATE OR REPLACE TABLE `SEASON` (
    `code_season` INT NOT NULL AUTO_INCREMENT UNIQUE,
    `number` SMALLINT NOT NULL,
    `episode_count` SMALLINT NOT NULL,
    `code_show` INT NOT NULL,
    PRIMARY KEY(`code_season`)
);
CREATE OR REPLACE TABLE `PERSON` (
    `code_person` INT NOT NULL AUTO_INCREMENT UNIQUE,
    `firstname` VARCHAR(255) NOT NULL,
    `lastname` VARCHAR(255) NOT NULL,
    PRIMARY KEY(`code_person`)
);
CREATE OR REPLACE TABLE `CASTING` (
    `code_casting` INT NOT NULL AUTO_INCREMENT UNIQUE,
    `role` VARCHAR(255) NOT NULL,
    `credit_order` INT,
    `code_show` INT NOT NULL,
    `code_person` INT NOT NULL,
    PRIMARY KEY(`code_casting`)
);
```

```

CREATE OR REPLACE TABLE `USER` (
    `code_user` INT NOT NULL AUTO_INCREMENT UNIQUE,
    `pseudo` VARCHAR(255) NOT NULL,
    `email` VARCHAR(255) NOT NULL,
    `role` VARCHAR(255) NOT NULL,
    `password` VARCHAR(255) NOT NULL,
    PRIMARY KEY(`code_user`)
);
CREATE OR REPLACE TABLE `FAVORITE` (
    `code_favorite` INT NOT NULL AUTO_INCREMENT UNIQUE,
    `code_user` INT NOT NULL,
    `code_show` INT NOT NULL,
    PRIMARY KEY(`code_favorite`)
);
ALTER TABLE `SHOW`
ADD FOREIGN KEY(`code_genre`) REFERENCES `GENRE`(`code_genre`)
ON UPDATE NO ACTION ON DELETE NO ACTION;
ALTER TABLE `SEASON`
ADD FOREIGN KEY(`code_show`) REFERENCES `SHOW`(`code_show`)
ON UPDATE NO ACTION ON DELETE NO ACTION;
ALTER TABLE `CASTING`
ADD FOREIGN KEY(`code_show`) REFERENCES `SHOW`(`code_show`)
ON UPDATE NO ACTION ON DELETE NO ACTION;
ALTER TABLE `CASTING`
ADD FOREIGN KEY(`code_person`) REFERENCES `PERSON`(`code_person`)
ON UPDATE NO ACTION ON DELETE NO ACTION;
ALTER TABLE `FAVORITE`
ADD FOREIGN KEY(`code_user`) REFERENCES `USER`(`code_user`)
ON UPDATE NO ACTION ON DELETE NO ACTION;
ALTER TABLE `FAVORITE`
ADD FOREIGN KEY(`code_show`) REFERENCES `SHOW`(`code_show`)
ON UPDATE NO ACTION ON DELETE NO ACTION;

```

Le code SQL fournit montre deux grandes étapes dans la création de la base de données :

1. Dans un premier temps, on note la création des tables avec **CREATE OR REPLACE TABLE**.
2. Dans un second temps, les clés étrangères (**FOREIGN KEY**) sont créées en altérant les tables précédemment créées avec **ALTER TABLE**.

## 2. Précisez les moyens utilisés :

Pour réaliser les différents schémas et tableaux de cette compétence professionnelle, j'ai utilisé :

- [MOCODO Online](#) pour réaliser le Modèle Conceptuel de Données
- [drawDB](#) pour réaliser le Modèle Physique de Données
- Le site [SQL.sh](#) pour bien comprendre le code SQL délivré par mon MPD

## 3. Avec qui avez-vous travaillé ?

J'ai travaillé seul sur ce projet réalisé lors de mon cursus de formation chez O'clock.



Liberté • Égalité • Fraternité

RÉPUBLIQUE FRANÇAISE

MINISTÈRE CHARGÉ  
DE L'EMPLOI

# DOSSIER PROFESSIONNEL (DP)

## 4. Contexte

Nom de l'entreprise, organisme ou association ➔ **Ecole O'clock**

Chantier, atelier, service ➔ **Projet réalisé au cours de la formation O'clock.**

Période d'exercice ➔ Du : **30/10/2023** au : **22/04/2024**

## 5. Informations complémentaires (facultatif)

## Activité-type 2

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

### CP 6 - Développer des composants d'accès aux données SQL et NoSQL

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

Dans le cadre de notre cursus de formation, nous avons créé le backoffice d'un site de vente de chaussures : oShop. Ce projet, codé en langage PHP, repose sur le modèle Modèle - Vue - Contrôleur (ou MVC) et nous a permis d'apprendre à accéder aux données de notre base de données.

#### L'accès à la base de données

Pour accéder à la base de données, j'ai créé un fichier **Database.php** conçu selon le patron de conception (ou design pattern) **Singleton**.

*"Singleton est un patron de conception de création qui garantit que l'instance d'une classe n'existe qu'en un seul exemplaire, tout en fournissant un point d'accès global à cette instance."* ([Refactoring Guru](#))

```
<?php

namespace App\Utils;

use PDO;

class Database
{
    private $dbh;

    private static $instance;

    private function __construct()
    {
        $configData = parse_ini_file(__DIR__ . '/../config.ini');

        try {
            $this->dbh = new PDO(
                "mysql:host={$configData['DB_HOST']};dbname={$configData['DB_NAME']};charset=utf8",
                $configData['DB_USERNAME'],
                $configData['DB_PASSWORD'],
                array(PDO::ATTR_ERRMODE => PDO::ERRMODE_WARNING)
            );
        } catch (\Exception $exception) {
            echo 'Erreur de connexion...<br>';
            echo $exception->getMessage() . '<br>';
            echo '<pre>';
            echo $exception->getTraceAsString();
            echo '</pre>';
            exit;
        }
    }

    public static function getPDO()
    {
        if (empty(self::$instance)) {
            self::$instance = new Database();
        }

        return self::$instance->dbh;
    }
}
```



# DOSSIER PROFESSIONNEL (DP)

MINISTÈRE CHARGÉ  
DE L'EMPLOI

Ce fichier **Database.php** a pour but de garantir la connexion à notre base de données quand on utilise **Database::getPDO()**. Passons en revue le code de ce fichier pour comprendre comment cela permet de nous connecter à la base de données.

**private \$dbh** est une propriété permettant de stocker un objet PDO qui représente la connexion à la base de données.

C'est d'ailleurs le but de cette classe native de PHP : en PHP, PDO (pour PHP Data Object) est une extension de PHP qui fournit une interface pour travailler et accéder à des bases de données.

**private static \$instance** est une propriété statique (uniquement liée à cette classe) qui va permettre de stocker l'unique instance de la classe Database.

**private function \_\_construct()** est un constructeur privé, c'est-à-dire que seul le code de la classe Database peut créer une instance de cette classe.

Tout d'abord, le constructeur récupère les données du fichier **config.ini** (capture ci-dessous) où sont stockées nos informations de connexion à la base de données MariaDB.

```
● ○ ●  
  
// Connexion base de données  
DB_HOST=127.0.0.1  
DB_NAME=oishop  
DB_USERNAME=jelly_oishop  
DB_PASSWORD=admin
```

Ensuite PHP essaie d'exécuter le code à l'intérieur du bloc "try", c'est là que sont utilisées les données du fichier **config.ini**. Si une erreur intervient, PHP exécute le bloc "catch" et affiche le code que l'on souhaite afficher, ici un message d'erreur.

**public static getPDO()** est une méthode statique qui va permettre de récupérer l'objet PDO représentant la connexion à la base de données. Cette méthode garantit la connexion active à la base de données à tout moment dans l'application. Si une connexion n'existe pas encore, elle est créée, sinon la connexion déjà existante est retournée.

Ce fichier **Database.php** va donc pouvoir me servir dans mes "Modèles" pour créer des méthodes qui vont permettre d'exécuter des requêtes SQL vers ma base de données. Ces méthodes pourront ensuite être utilisées par les "Contrôleurs" pour envoyer des données dans les "Vues". Cela permet également d'éviter de réécrire l'accès à la base de données avant chaque requête SQL dans les "Modèles".

## Effectuer des requêtes

Une fois la connexion à la base de données effectuée, je peux effectuer des requêtes SQL dans mes "Modèles" pour récupérer les données de la base de données qui m'intéressent avant de pouvoir les envoyer à la "Vue" via le "Contrôleur".

Prenons un exemple pour expliquer cela concrètement :

#	Nom	Image	Description		
1	Kissing		Proinde concepta rabie saeviore, quam desperatio incendebat et fames, amplificatis viribus ardore incohibili in excidium urbium matris Seleuciae efferebantur, quam comes tuebatur Castricius tresque legiones bellicis sudoribus induratae.	<input checked="" type="checkbox"/>	
2	Pink Lady		Nunc vero inanes flatus quorundam vile esse quicquid extra urbis pomerium nascitur aestimant praeter orbos et caelibes, nec credi potest qua obsequiorum diversitate coluntur homines sine liberis Romae.	<input checked="" type="checkbox"/>	
3	Panda		Homines enim eruditos et sobrios ut infaustos et inutiles vitant, eo quoque accedente quod et nomenclatores adsueti haec et talia venditare, mercede accepta lucris quosdam et prandis inserunt subditios ignobiles et obscuros.	<input checked="" type="checkbox"/>	
4	Zombie		Sed si ille hac tam eximia fortuna propter utilitatem rei publicae frui non properat, ut omnia illa conficiat, quid ego, senator, facere debeo, quem, etiamsi ille aliud vellet, rei publicae consulere oportet?	<input checked="" type="checkbox"/>	
5	Minion		Ut enim benefici liberalesque sumus, non ut exigamus gratiam (neque enim beneficium faeneramus sed natura propensi ad liberalitatem sumus), sic amicitiam non spe mercedis adducti sed quod omnis eius fructus in ipso amore inest, expetendam putamus.	<input checked="" type="checkbox"/>	
6	Père Noël		Tempore quo primis auspiciis in mundanum fulgorem surgeret victura dum erunt homines Roma, ut augeretur sublimibus incrementis, foedere pacis aeternae Virtus convenit atque Fortuna plerumque dissidentes, quarum si altera defuisse, ad perfectam non venerat summittat.	<input checked="" type="checkbox"/>	

Cette page du backoffice du projet oShop détaille la liste de tous les produits disponibles sur le site. Essayons de retracer le chemin de la requête SQL qui a permis d'afficher tous ces produits.

**1ère étape :** Création de la méthode **findAll()** dans le Modèle **Product.php**

```
/** * Méthode permettant de récupérer tous les enregistrements de la table product */ * @return Product[] */ public static function findAll():array { $pdo = Database::getPDO(); $sql = 'SELECT * FROM `product`'; $pdoStatement = $pdo->query($sql); $results = $pdoStatement->fetchAll(PDO::FETCH_CLASS, self::class); return $results; }
```

On crée une méthode qui va nous permettre de récupérer tous les produits de la base de données :  
\$sql = 'SELECT \* FROM product';

La méthode **fetchAll()** permet de retourner les résultats sous forme d'objets de la classe Product.

Enfin, la méthode **findAll()** retourne un tableau avec tous les objets de la classe Product.



# DOSSIER PROFESSIONNEL (DP)

**2ème étape :** Préparer les données dans le “Contrôleur” pour les envoyer à la “Vue”

```
● ● ●

public function browseAction()
{
    // 1. On prépare les données
    $allProductList = Product::findAll();
    $dataForView = [
        'productList' => $allProductList,
    ];

    // 2. On appelle la vue
    $this->show('product/browse', $dataForView);
}
```

Dans le “Contrôleur” **ProductController.php**, nous créons une méthode **browseAction()** qui va permettre de préparer les données : **\$allProductList = Product::findAll();**

La variable **\$allProductList** appelle la méthode **findAll()** de la classe Product. La variable **\$allProductList** contiendra un tableau d’objets de la classe Product.

La variable **\$dataForView** est un tableau associatif dont la clé ‘**productList**’ a pour valeur le tableau de tous les produits récupérés à partir de la base de données.

Une fois les données préparées, le “Contrôleur” va appeler la “Vue” et lui transmettre le tableau **\$dataForView**.

Cette dernière ligne de la méthode **browseAction()** appelle la méthode **show(string \$viewName, \$viewData = [])** du fichier **CoreController.php** (ce fichier possède toutes les méthodes communes à l’ensemble des “Contrôleurs”). On voit que la méthode **show()** passe en arguments, le chemin vers le fichier de la “Vue” à afficher ainsi qu’un tableau de données à transmettre à la “Vue” et c’est ce que l’on voit en lisant la ligne de code : **\$this->show('product/browse', \$dataForView);**

**3ème étape :** Afficher les données dans la “Vue”

Une fois que les données sont transmises à la “Vue”, il faut pouvoir les afficher dans celle-ci. C'est ici que va se jouer la dynamisation du backoffice avec le PHP.

La “Vue” **product/browse.tpl.php** n'est autre que le bloc situé entre le header et le footer de la page que l'on souhaite afficher.

```
require_once __DIR__ . '/../../Views/layout/header.tpl.php';
require_once __DIR__ . '/../../Views/' . $viewName . '.tpl.php';
require_once __DIR__ . '/../../Views/layout/footer.tpl.php';
```

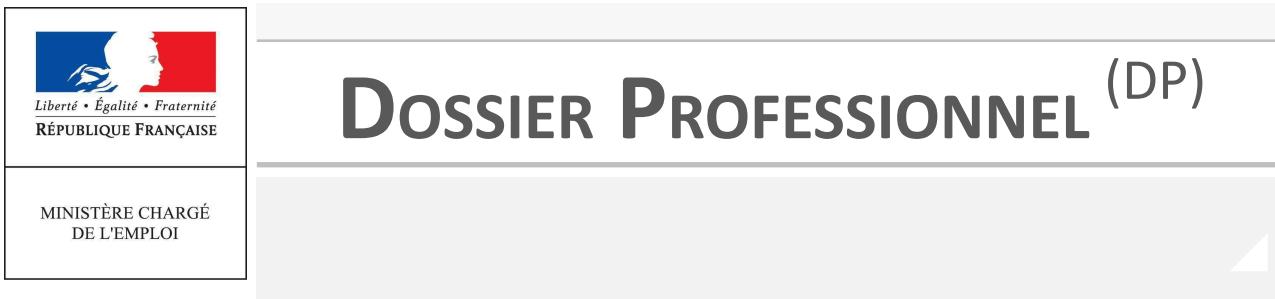
En effet, quand on regarde la méthode **show(string \$viewName, \$viewData = [])** du fichier **CoreController.php**, cette dernière demande d'afficher les “Vues” suivantes et on constate que cela est dynamisé grâce à la variable **\$viewName** qui va venir afficher la bonne “Vue” entre le header et le footer.

Voici donc la “Vue” **product/browse.tpl.php** qui va permettre d'afficher les données transmises par la base de données :

```
<a href="= $router-&gt;generate('product-add'); ?" class="btn btn-success float-end">Ajouter</a>
<h2>Liste des produits</h2>
<table class="table table-hover mt-4">
    <thead>
        <tr>
            <th scope="col">#</th>
            <th scope="col">Nom</th>
            <th scope="col">Image</th>
            <th scope="col">Description</th>
            <th scope="col"></th>
        </tr>
    </thead>
    <tbody>
        <?php foreach ($productList as $currentProduct) : ?>
        <tr>
            <th scope="row">= $currentProduct-&gt;getId(); ?&gt;&lt;/th&gt;
            &lt;td&gt;<?= $currentProduct-&gt;getName(); ?&gt;&lt;/td&gt;
            &lt;td&gt;&lt;img width="100px;" src="<?= $currentProduct-&gt;getPicture(); ?&gt;" /&gt;&lt;/td&gt;
            &lt;td&gt;<?= $currentProduct-&gt;getDescription(); ?&gt;&lt;/td&gt;
            &lt;td class="text-end"&gt;
                &lt;a href="<?= $router-&gt;generate('product-update', ['id' =&gt; $currentProduct-&gt;getId()]); ?&gt;" class="btn btn-sm btn-warning"&gt;
                    &lt;i class="fa fa-pencil-square-o" aria-hidden="true"&gt;&lt;/i&gt;
                &lt;/a&gt;
                &lt;!-- Example single danger button --&gt;
                &lt;div class="btn-group"&gt;
                    &lt;button type="button" class="btn btn-sm btn-danger dropdown-toggle"
                        data-bs-toggle="dropdown" aria-haspopup="true" aria-expanded="false"&gt;
                        &lt;i class="fa fa-trash-o" aria-hidden="true"&gt;&lt;/i&gt;
                    &lt;/button&gt;
                    &lt;div class="dropdown-menu"&gt;
                        &lt;a class="dropdown-item" href="<?= $router-&gt;generate('product-delete', ['id' =&gt; $currentProduct-&gt;getId()]); ?&gt;"&gt;Oui, je veux supprimer&lt;/a&gt;
                        &lt;a class="dropdown-item" href="#" data-toggle="dropdown"&gt;Oups !&lt;/a&gt;
                    &lt;/div&gt;
                &lt;/div&gt;
            &lt;/td&gt;
        &lt;/tr&gt;
        &lt;?php endforeach; ?&gt;
    &lt;/tbody&gt;
&lt;/table&gt;</pre
```

C'est la boucle **foreach** qui va venir dynamiser cette “Vue” en créant une nouvelle ligne du tableau par produits existants dans le tableau d'objets transmis :

**<?php foreach (\$productList as \$currentProduct) : ?>**



# DOSSIER PROFESSIONNEL (DP)

En bouclant sur chaque objet du tableau transmis, cela permet d'afficher, pour chaque produit :

- son identifiant <?= \$currentProduct->getId(); ?>,
- son nom <?= \$currentProduct->getName(); ?>,
- son image <?= \$currentProduct->getPicture(); ?>,
- sa description <?= \$currentProduct->getDescription(); ?>

Nous pouvons donc afficher sur une seule page la totalité des produits de notre base de données sans avoir besoin de créer une ligne par produits manuellement, PHP et le pattern MVC nous permettent de dynamiser la page facilement et d'afficher la liste que nous pouvons voir sur la capture d'écran de la page 34.

## Exemple de jointure

Dans l'exemple que nous venons de voir, nous sélectionnons tous les items de la table Product cependant, il convient de penser à des requêtes qui peuvent être plus spécifiques. Par exemple, si je crée une page où je ne veux afficher que les produits de la marque PHPieds :

Sélectionner: brand					Table: product																																																	
<a href="#">Afficher les données</a>		<a href="#">Afficher la structure</a>		<a href="#">Modifier la table</a>	<a href="#">Afficher les données</a>		<a href="#">Afficher la structure</a>		<a href="#">Modifier la table</a>																																													
<a href="#">Sélectionner</a>		<a href="#">Rechercher</a>		<a href="#">Trier</a>	Limite 50		Longueur 100																																															
SELECT * FROM `brand` LIMIT 50 (0.001 s) <a href="#">Modifier</a>																																																						
<table border="1"> <thead> <tr> <th><input type="checkbox"/> Modification</th> <th><a href="#">id</a></th> <th><a href="#">name</a></th> <th><a href="#">created_at</a></th> <th><a href="#">updated_at</a></th> </tr> </thead> <tbody> <tr><td><input type="checkbox"/> modifier</td><td>1</td><td>oCirage</td><td>2018-10-17 09:00:00</td><td>NULL</td></tr> <tr><td><input type="checkbox"/> modifier</td><td>2</td><td>BOOTstrap</td><td>2018-10-17 09:00:00</td><td>NULL</td></tr> <tr><td><input type="checkbox"/> modifier</td><td>3</td><td>Talonette</td><td>2018-10-17 09:00:00</td><td>NULL</td></tr> <tr><td><input type="checkbox"/> modifier</td><td>4</td><td>Shossures</td><td>2018-10-17 09:00:00</td><td>NULL</td></tr> <tr><td><input type="checkbox"/> modifier</td><td>5</td><td>O'shoes</td><td>2018-10-17 09:00:00</td><td>NULL</td></tr> <tr><td><input type="checkbox"/> modifier</td><td>6</td><td>Pattes d'eph</td><td>2018-10-17 09:00:00</td><td>NULL</td></tr> <tr><td><input type="checkbox"/> modifier</td><td>7</td><td>PHPieds</td><td>2018-10-17 09:00:00</td><td>NULL</td></tr> <tr><td><input type="checkbox"/> modifier</td><td>8</td><td>oPompes</td><td>2018-10-17 09:00:00</td><td>NULL</td></tr> </tbody> </table>										<input type="checkbox"/> Modification	<a href="#">id</a>	<a href="#">name</a>	<a href="#">created_at</a>	<a href="#">updated_at</a>	<input type="checkbox"/> modifier	1	oCirage	2018-10-17 09:00:00	NULL	<input type="checkbox"/> modifier	2	BOOTstrap	2018-10-17 09:00:00	NULL	<input type="checkbox"/> modifier	3	Talonette	2018-10-17 09:00:00	NULL	<input type="checkbox"/> modifier	4	Shossures	2018-10-17 09:00:00	NULL	<input type="checkbox"/> modifier	5	O'shoes	2018-10-17 09:00:00	NULL	<input type="checkbox"/> modifier	6	Pattes d'eph	2018-10-17 09:00:00	NULL	<input type="checkbox"/> modifier	7	PHPieds	2018-10-17 09:00:00	NULL	<input type="checkbox"/> modifier	8	oPompes	2018-10-17 09:00:00	NULL
<input type="checkbox"/> Modification	<a href="#">id</a>	<a href="#">name</a>	<a href="#">created_at</a>	<a href="#">updated_at</a>																																																		
<input type="checkbox"/> modifier	1	oCirage	2018-10-17 09:00:00	NULL																																																		
<input type="checkbox"/> modifier	2	BOOTstrap	2018-10-17 09:00:00	NULL																																																		
<input type="checkbox"/> modifier	3	Talonette	2018-10-17 09:00:00	NULL																																																		
<input type="checkbox"/> modifier	4	Shossures	2018-10-17 09:00:00	NULL																																																		
<input type="checkbox"/> modifier	5	O'shoes	2018-10-17 09:00:00	NULL																																																		
<input type="checkbox"/> modifier	6	Pattes d'eph	2018-10-17 09:00:00	NULL																																																		
<input type="checkbox"/> modifier	7	PHPieds	2018-10-17 09:00:00	NULL																																																		
<input type="checkbox"/> modifier	8	oPompes	2018-10-17 09:00:00	NULL																																																		
<table border="1"> <thead> <tr> <th>Colonne</th> <th>Type</th> <th>Commentaire</th> </tr> </thead> <tbody> <tr><td><b>id</b></td><td>int(10) unsigned</td><td>Incrément automatique</td></tr> <tr><td><b>name</b></td><td>varchar(64)</td><td></td></tr> <tr><td><b>description</b></td><td>text NULL</td><td></td></tr> <tr><td><b>picture</b></td><td>varchar(255) NULL</td><td></td></tr> <tr><td><b>price</b></td><td>decimal(10,2) [0.00]</td><td></td></tr> <tr><td><b>rate</b></td><td>tinyint(1) [0]</td><td></td></tr> <tr><td><b>status</b></td><td>tinyint(1) [0]</td><td></td></tr> <tr><td><b>created_at</b></td><td>timestamp [<b>current_timestamp()</b>]</td><td></td></tr> <tr><td><b>updated_at</b></td><td>timestamp NULL</td><td></td></tr> <tr><td><b>brand_id</b></td><td>int(10) unsigned</td><td></td></tr> <tr><td><b>category_id</b></td><td>int(10) unsigned NULL</td><td></td></tr> <tr><td><b>type_id</b></td><td>int(11)</td><td></td></tr> </tbody> </table>										Colonne	Type	Commentaire	<b>id</b>	int(10) unsigned	Incrément automatique	<b>name</b>	varchar(64)		<b>description</b>	text NULL		<b>picture</b>	varchar(255) NULL		<b>price</b>	decimal(10,2) [0.00]		<b>rate</b>	tinyint(1) [0]		<b>status</b>	tinyint(1) [0]		<b>created_at</b>	timestamp [ <b>current_timestamp()</b> ]		<b>updated_at</b>	timestamp NULL		<b>brand_id</b>	int(10) unsigned		<b>category_id</b>	int(10) unsigned NULL		<b>type_id</b>	int(11)							
Colonne	Type	Commentaire																																																				
<b>id</b>	int(10) unsigned	Incrément automatique																																																				
<b>name</b>	varchar(64)																																																					
<b>description</b>	text NULL																																																					
<b>picture</b>	varchar(255) NULL																																																					
<b>price</b>	decimal(10,2) [0.00]																																																					
<b>rate</b>	tinyint(1) [0]																																																					
<b>status</b>	tinyint(1) [0]																																																					
<b>created_at</b>	timestamp [ <b>current_timestamp()</b> ]																																																					
<b>updated_at</b>	timestamp NULL																																																					
<b>brand_id</b>	int(10) unsigned																																																					
<b>category_id</b>	int(10) unsigned NULL																																																					
<b>type_id</b>	int(11)																																																					

Comme ma base de données a été pensé en ce sens lors de sa conception, j'ai bien une clé étrangère **brand\_id** dans ma table **product**, ce qui va me permettre d'effectuer une jointure. Une jointure SQL est une opération qui va permettre de combiner les lignes de deux tables différentes en fonction d'une relation existante entre une ou des colonnes de ces tables.

Je peux donc écrire ma jointure (je n'ai volontairement sélectionné que les trois colonnes id, name et brand\_id pour que tout passe dans la capture écran)...

```
SELECT product.id, product.name, product.brand_id
FROM product
INNER JOIN brand ON product.brand_id = brand.id
WHERE brand.name = 'PHPieds';
```

... et la tester dans l'outil Requête SQL d'Adminer :

### Requête SQL

```
SELECT product.id, product.name, product.brand_id
FROM product
INNER JOIN brand ON product.brand_id = brand.id
WHERE brand.name = 'PHPieds'
```

<b>id</b>	<b>name</b>	<b>brand_id</b>
4	Zombie	7
11	Teeturtle	7
14	Shark	7
23	Jardon	7
25	Sans dale	7

5 lignes (0.002 s) [Modifier](#), [Explain](#), [Exporter](#)

```
SELECT product.id, product.name, product.brand_id
FROM product
INNER JOIN brand ON product.brand_id = brand.id
WHERE brand.name = 'PHPieds';
```

Je vois donc que ma jointure SQL fonctionne car elle ne me ressort bien que les produits dont la **brand\_id** est 7, ce qui correspond bien à la **brand** PHPieds.

On pourrait donc imaginer la création d'une méthode **findByBrand(int \$brandId)** dans notre **ProductController.php** qui irait chercher les produits d'une marque spécifique.

#### Interagir avec la base de données

Les requêtes SQL peuvent également nous permettre de modifier la base de données. Imaginons que je veuille supprimer un produit de la base de données car je ne vais finalement pas commercialiser le produit en question, comment faire ?

Pour cela, je vais créer une méthode **delete(int \$id):bool** dans notre “Modèle” Product qui va prendre en paramètre l'**\$id** du produit que je veux supprimer. La méthode retournera **true** si la requête réussit et **false** si la requête échoue.



# DOSSIER PROFESSIONNEL (DP)

MINISTÈRE CHARGÉ  
DE L'EMPLOI



```
public static function delete(int $id):bool
{
    $pdo = Database::getPDO();
    $sql = "DELETE FROM `product` WHERE id = :id;";

    $pdoStatement = $pdo->prepare($sql);

    $deleteIsOk = $pdoStatement->execute(['id' => $id]);

    return $deleteIsOk;
}
```

Voici la méthode **delete()** de mon “Modèle” Product, expliquons brièvement ce qu’elle fait :

- Dans un premier temps, elle obtient une instance de la classe PDO en appelant la méthode statique **getPDO()** de la classe Database que nous avons détaillée précédemment. L’instance est stockée dans la variable **\$pdo**.
- Ensuite, la variable **\$sql** nous permet de stocker la requête SQL que l’on souhaite effectuer, à savoir : **"DELETE FROM `product` WHERE id = :id;"**. Ici la requête a pour but de supprimer une ligne de la table **product** où id correspond à la valeur de **:id**. J’utilise **:id** dans ma requête SQL car cela va me permettre d’utiliser ce qu’on appelle une requête préparée qui va être utile pour se protéger d’éventuelles attaques par injection SQL.
- La requête SQL est ensuite préparée, grâce à la méthode **prepare()** de l’objet PDO et stockée dans la variable **\$pdoStatement**. À noter, la méthode **prepare()** retourne un objet **PDOStatement** qui peut être utilisé pour exécuter la requête en prenant des valeurs en paramètres.
- Enfin, la variable **\$deleteIsOk** stocke le résultat, **true** ou **false**, de l’exécution de la requête préparée, requête qui a en paramètre l’**id** du produit à supprimer. La méthode **execute()** exécute la requête préparée avec les valeurs de paramètres spécifiées dans un tableau associatif.
- La fonction retourne le résultat **true** ou **false** de l’exécution de la requête SQL.

En développement, quand on développe des méthodes qui permettent d’interagir avec la base de données, on dit qu’on met en place un CRUD. CRUD est l’acronyme permettant de désigner les quatre opérations de base pour la persistance des données : **Create, Read, Update, Delete**.

À travers les exemples de cette Compétence Professionnelle, nous avons détaillé les méthodes permettant d’afficher, **findAll()**, et de supprimer, **delete()**, un produit dans notre base de données,

mais nous avions également mis en place les méthodes **insert()** et **update()** dans notre projet oShop afin de pouvoir créer ou mettre à jour des produits.

## 2. Précisez les moyens utilisés :

Pour réaliser ce projet du cursus O'clock, j'ai utilisé l'éditeur de code Visual Studio Code.

Pour tester l'affichage de la page web, j'ai utilisé le navigateur web Google Chrome et notamment ses outils d'aide au développement pour faciliter l'intégration et le débogage de mon code.

J'ai également utilisé la documentation en ligne de PHP afin de comprendre au mieux la syntaxe de base de PHP (types, variables, opérateurs, fonctions, classe et objets, etc...).

- <https://www.php.net/manual/fr/>

Pour tester mes requêtes SQL, j'ai utilisé l'outil Requête SQL d'Adminer.

## 3. Avec qui avez-vous travaillé ?

J'ai travaillé seul sur ce projet réalisé lors de mon cursus de formation chez O'clock.

## 4. Contexte

Nom de l'entreprise, organisme ou association ▶ **Ecole O'clock**

Chantier, atelier, service ▶ **Projet réalisé au cours de la formation O'clock.**

Période d'exercice ▶ Du : **30/10/2023** au : **22/04/2024**

## 5. Informations complémentaires (facultatif)



MINISTÈRE CHARGÉ  
DE L'EMPLOI

# DOSSIER PROFESSIONNEL (DP)

## Activité-type 2

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

### CP 7 - *Développer des composants métier côté serveur*

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

Pour cette Compétence Professionnelle, je vais reprendre l'exemple du projet O'flix que j'ai déjà utilisé dans la CP 5. Pour rappel, O'flix est une base de données de films et séries semblable à un site comme [IMDB](#) :

- les utilisateurs peuvent se connecter et mettre des films/séries en favoris, noter et laisser un commentaire sur un film ou une série,
- les administrateurs ont accès à un backoffice qui permet de créer un nouveau film ou une nouvelle série.

Dans ce projet, j'ai créé un certain nombre de composants métier côté serveur. Pour rappel, les composants métier sont des règles et des processus qui déterminent comment fonctionne le site O'flix et comment il traite les données.

Pour illustrer cette CP, nous prendrons deux exemples de composants métier côté serveur créés pour notre site O'flix :

1. Le système d'authentification permettant de vérifier les informations d'identification des utilisateurs et de déterminer les actions qu'ils sont autorisés à effectuer sur le site.
2. Le système de commentaires qui permet aux utilisateurs de laisser des critiques sur les films et/ou les séries qu'ils ont vu(e)s.

**Exemple 1 :** Mise en place d'un système d'authentification et d'autorisation des utilisateurs

Lors de la création de notre projet O'flix sur Symfony, j'ai mis en place un système de gestion des utilisateurs. En se connectant, le composant analyse les autorisations de l'utilisateur et, en fonction de son rôle, il pourra avoir accès ou non à certaines parties du site.

Les rôles mis en place sur ce projet étaient :

- **ROLE\_USER** : accès aux fonctionnalités du site (écrire des reviews, mettre des films/séries en favoris)

- **ROLE\_MANAGER** : accès aux fonctionnalités du site (écrire des reviews, mettre des films/séries en favoris) et accès au backoffice uniquement pour voir la liste des films/séries
- **ROLE\_ADMIN** : accès aux fonctionnalités du site (écrire des reviews, mettre des films/séries en favoris) et accès au backoffice du site pour gérer les films et séries et les utilisateurs

Pour arriver à ce résultat, j'ai dû procéder par étape :

Dans un premier temps, j'ai utilisé le [Security Bundle de Symfony](#), qui fournit des fonctionnalités de sécurité pour les applications Symfony. Tout d'abord, j'ai créé une classe User à l'aide de la commande **php bin/console make:user** ce qui va me permettre d'avoir une classe contenant des propriétés comme email, rôles et mot de passe. Cette classe permet simplement de représenter un utilisateur du site O'flix.

Une fois la classe User créée, je configure le système de sécurité de Symfony pour utiliser la classe User. Cette configuration se fait dans le fichier **security.yaml** :



```

security:
    # https://symfony.com/doc/current/security.html#registering-the-user-hashing-passwords
    password_hashers:
        Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface: 'auto'
    # https://symfony.com/doc/current/security.html#loading-the-user-the-user-provider
    providers:
        # used to reload user from session & other features (e.g. switch_user)
        app_user_provider:
            entity:
                class: App\Entity\User
                property: email
    firewalls:
        login:
            pattern: ^/api/login
            stateless: true
            json_login:
                check_path: /api/login_check
                success_handler: lexik_jwt_authentication.handler.authentication_success
                failure_handler: lexik_jwt_authentication.handler.authentication_failure
        api:
            pattern: ^/api
            stateless: true
            jwt: ~
        dev:
            pattern: ^/((profiler|wdt)|css|images|js)/
            security: false
        main:
            lazy: true
            provider: app_user_provider
            form_login:
                login_path: app_login
                check_path: app_login
                enable.csrf: true
            logout:
                path: app_logout

    # Easy way to control access for large sections of your site
    # Note: Only the *first* access control that matches will be used
    access_control:
        - { path: ^/review, roles: ROLE_USER }
        - { path: ^/favorites, roles: ROLE_USER }
        - { path: ^/admin/movie/add, roles: ROLE_ADMIN }
        - { path: ^/admin/movie/update, roles: ROLE_ADMIN }
        - { path: ^/admin/movie/remove, roles: ROLE_ADMIN }
        - { path: ^/admin/movie, roles: [ROLE_MANAGER, ROLE_ADMIN] }

    when@test:
        security:
            password_hashers:
                # By default, password hashers are resource intensive and take time. This is
                # important to generate secure password hashes. In tests however, secure hashes
                # are not important, waste resources and increase test times. The following
                # reduces the work factor to the lowest possible values.
                Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface:
                    algorithm: auto
                    cost: 4 # Lowest possible value for bcrypt
                    time_cost: 3 # Lowest possible value for argon
                    memory_cost: 10 # Lowest possible value for argon

```



# DOSSIER PROFESSIONNEL (DP)

MINISTÈRE CHARGÉ  
DE L'EMPLOI

Dans ce fichier, voici le rôle des différentes sections :

- **password\_hashers** configure les hashers de mot de passe utilisés pour sécuriser les mots de passe des utilisateurs
- **providers** permet de définir les fournisseurs de données utilisés pour charger les utilisateurs, ici c'est la classe User qui va permettre d'identifier un utilisateur en se basant sur la propriété email
- **firewalls** permet de configurer les pare-feux de sécurité pour le site
- **access\_control** permet de contrôler quels rôles ont accès aux différentes parties de notre site
- **when@test** contient des configurations spécifiques qui sont appliquées lors de l'exécution de tests

Une fois le fichier **security.yaml** configuré, je peux gérer l'authentification et les autorisations des utilisateurs du site O'flix. Quand un utilisateur se connecte, Symfony va utiliser les informations d'identification fournies par l'utilisateur (son email et son mot de passe dans notre cas) pour vérifier son identité.

The screenshot shows the O'flix login interface. At the top, there is a navigation bar with a movie popcorn icon, the text "O'FLIX", "Accueil", "Films, séries TV", "Ma liste", "Connexion", a search bar with placeholder "Rechercher...", and a magnifying glass icon. Below the navigation, a pink error message box contains the text "Identifiants invalides." In the center, the heading "Se connecter" is displayed above two input fields: "Email" containing "pierre@oflix.com" and "Mot de passe" containing "\*\*\*\*\*". A blue "Se connecter" button is positioned below these fields. At the bottom of the page, there is footer text: "O'flix made with ❤ at O'clock !", "Thème : 🎥 Netflix", and "Film aléatoire : [Moonrise Kingdom](#)".

Si les identifiants sont mauvais, un message d'erreur apparaît pour prévenir l'utilisateur qu'il y a une erreur dans les informations qu'il a soumises au formulaire de connexion.



## Films, séries TV et popcorn en illimité.

Où que vous soyez. Gratuit pour toujours.

Si les identifiants sont corrects, l'utilisateur peut voir qu'il est bien connecté car son nom s'affiche à côté du champ de recherche (rectangle bleu sur la capture d'écran ci-dessus). Symfony authentifie alors l'utilisateur et lui donne accès aux parties qu'il peut visiter.

Ici, l'utilisateur **pierre@oflix.com** à le rôle **ROLE\_ADMIN**.

```
● ● ●

# Easy way to control access for large sections of your site
# Note: Only the *first* access control that matches will be used
access_control:
    - { path: ^/review, roles: ROLE_USER }
    - { path: ^/favorites, roles: ROLE_USER }
    - { path: ^/admin/movie/add, roles: ROLE_ADMIN }
    - { path: ^/admin/movie/update, roles: ROLE_ADMIN }
    - { path: ^/admin/movie/remove, roles: ROLE_ADMIN }
    - { path: ^/admin/movie, roles: [ROLE_MANAGER, ROLE_ADMIN] }
```

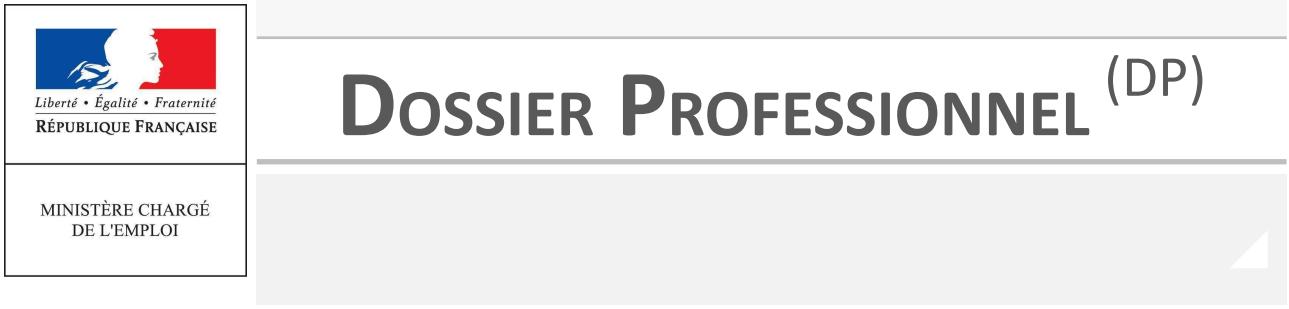
Grâce à la section **access\_control** du fichier **security.yaml**, on voit que Pierre a bien accès à l'ensemble des fonctionnalités du site (ainsi qu'au backoffice et à la gestion du backoffice).

Je n'ai pas défini de hiérarchie des rôles dans mon fichier **security.yaml**. Cependant, ma classe User a une logique qui permet de faire bénéficier, par défaut, du rôle User à tous les utilisateurs. Pierre a donc, en réalité, deux rôles : **ROLE\_USER & ROLE\_ADMIN**.

```
● ● ●

/**
 * @see UserInterface
 */
public function getRoles(): array
{
    $roles = $this->roles;
    // guarantee every user at least has ROLE_USER
    $roles[] = 'ROLE_USER';

    return array_unique($roles);
}
```



### Security

Token Firewall Listeners Authenticators Access Decision

Username	pierre@oflix.com	Authenticated	
Property	Value		
Roles	[ "ROLE_ADMIN" "ROLE_USER" ]		
Inherited Roles	none		
Token	Symfony\Component\Security\Core\Authentication\Token\UsernamePasswordToken {#1858 ▾ -user: App\Entity\User {#1901 ...} -roleNames: [ ]-attributes: [] -firewallName: "main" }		

Et c'est ce que nous confirme le Symfony Profiler quand nous sommes connectés avec **pierre@oflix.com**. Pierre hérite donc, par défaut, des deux premières règles de la section **access\_control** du fichier **security.yaml**.

J'ai créé un autre utilisateur, **guillaume@oflix.com**, qui n'a que le rôle **ROLE\_USER**. Quand j'essaie d'aller avec ce compte sur l'adresse d'accueil du backoffice : **/admin/movie**, voici ce qui s'affiche :

Symfony Exception      Symfony Docs  
AccessDeniedException > AccessDeniedHttpException      HTTP 403 Forbidden

Access Denied.

Exceptions 2   Logs 1   Stack Traces 2

Symfony\Component\HttpFoundation\Exception\AccessDeniedHttpException  
AccessDeniedHttpException  
▶ Show exception properties

Symfony\Component\Security\Core\Exception\AccessDeniedException  
AccessDeniedException  
Access Denied.

J'ai une erreur 403, c'est-à-dire que le serveur comprend la requête mais ne peut pas l'afficher car l'utilisateur ne possède pas les droits d'accès.

Dans le backoffice, j'ai ensuite une section Users qui nous permet de gérer les utilisateurs. Dans mon projet, je n'avais pas défini qui avait accès à cette page dans la section **access\_control** sur le fichier **security.yaml**, mais j'aurais permis l'accès uniquement aux utilisateurs ayant le rôle **ROLE\_ADMIN**.

À partir de cette page du backoffice, il est possible de voir, créer, mettre à jour et supprimer un utilisateur.

The screenshot shows a user management interface for a platform named "BACK'O'FLIX". The top navigation bar includes links for "Accueil", "Films, séries TV", "Ma liste", "Users", and a user profile for "pierre@oflix.com ROLE\_ADMIN". A search bar and a magnifying glass icon are also present. The main content area is titled "Liste des utilisateurs" and displays a table with three rows of user data:

ID	Email	Roles	actions
1	pierre@oflix.com	["ROLE_ADMIN", "ROLE_USER"]	<a href="#">Voir</a> <a href="#">Éditer</a>
2	jessica@oflix.com	["ROLE_MANAGER", "ROLE_USER"]	<a href="#">Voir</a> <a href="#">Éditer</a>
3	guillaume@oflix.com	["ROLE_USER"]	<a href="#">Voir</a> <a href="#">Éditer</a>

Below the table is a link "[Créer un nouvel utilisateur](#)". At the bottom of the page, there is a footer note: "O'flix made with ❤ at O'clock ! Thème : 🎬 Netflix".

### Exemple 2 : Mise en place d'un système de commentaires

La mise en place d'un système de commentaires peut également être considéré comme un composant métier dans la mesure où cette fonctionnalité est une composante clé de l'expérience utilisateur. En soi, le site peut tout à fait fonctionner sans les commentaires, mais cela apporte un plus pour les utilisateurs qui vont pouvoir donner leur avis, débattre, etc...

La gestion de ces commentaires implique donc une logique métier spécifique puisqu'il va falloir pouvoir modérer ces commentaires pour éviter le spam ou les insultes par exemple.

De plus, du côté de la persistance des données, les commentaires doivent être stockés en base de données et cela implique de pouvoir les ajouter, les modifier ou les supprimer.

Enfin, les commentaires nécessitent une coordination avec différents composants : dans le cas de mon projet, il faut que je puisse lier un commentaire à un utilisateur et à un film ou une série.

Et concrètement, comment ai-je mis cela en place ? J'ai suivi la logique Modèle Vue Contrôleur du framework Symfony.

### Étape 1 : Création de l'entité, du modèle et du contrôleur

À l'aide de la console, j'ai utilisé la commande **php bin/console make:entity** proposée par le [Symfony MakerBundle](#) pour créer mon “Entité” dans le fichier **Review.php** (situé dans le dossier Entity). Le maker de Symfony permet ensuite de nommer son “Entité”, d'y ajouter ses propriétés et éventuellement de gérer les relations entre différentes “Entités”.

L'utilisation du **Symfony MakerBundle** pour créer une “Entité” permet de créer automatiquement, en parallèle, le “Modèle” dans le fichier **ReviewRepository.php** (situé dans le dossier Repository). C'est dans ce fichier que seront effectuées les requêtes SQL qui seront envoyées au “Contrôleur”.



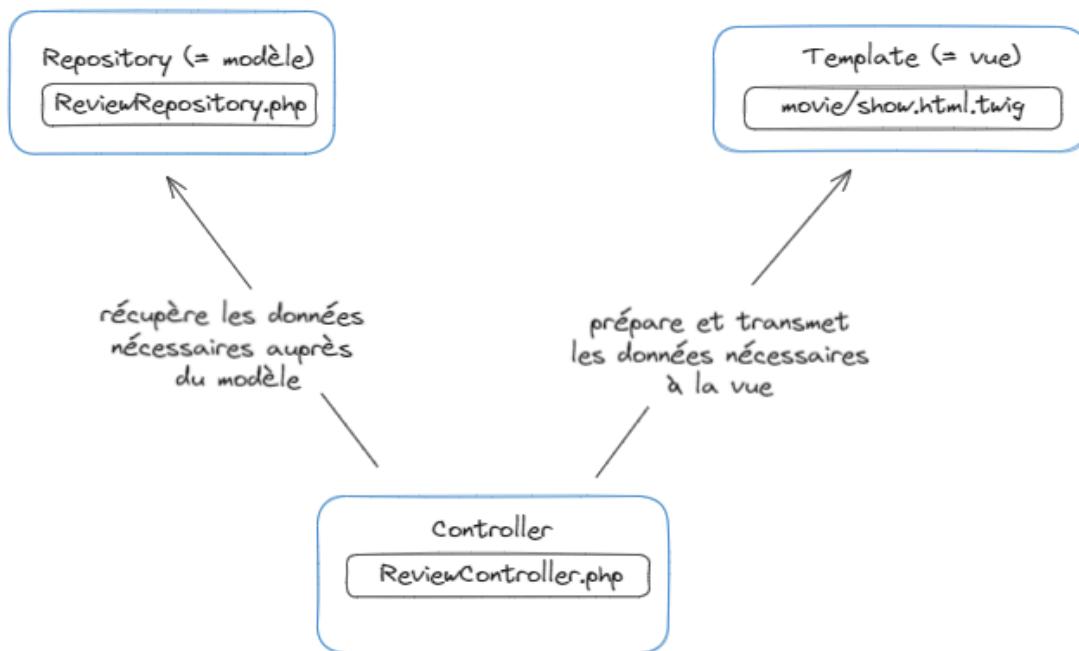
# DOSSIER PROFESSIONNEL (DP)

MINISTÈRE CHARGÉ  
DE L'EMPLOI

À l'aide de la console, j'ai utilisé la commande **php bin/console make:controller** proposée par le **Symfony MakerBundle** pour créer mon “Contrôleur” dans le fichier **ReviewController.php** (situé dans le dossier Controller).

C'est dans ce fichier que je vais préparer et transmettre les requêtes les données que je souhaite afficher sur mon template. Ici, je veux afficher les commentaires des utilisateurs sur la page d'un film ou d'une série, donc dans le fichier **movie/show.html.twig**.

Finalement, si je schématisé, cela donne :



## Étape 2 : Du côté de la Vue (template)

Du côté de l'affichage des commentaires, il faut donc aller dans le fichier **movie/show.html.twig** qui correspond au template d'affichage des détails d'une série ou d'un film, car c'est en bas de ce template que je souhaite afficher les commentaires.

Voici l'extrait de code qui permet l'affichage des commentaires :

```
● ● ●

<section>
    {% if show.reviews is not empty %}
        <h2 class="my-3">Les reviews</h2>
    {% else %}
        <h2 class="my-3">Pas encore de reviews</h2>
    {% endif %}
    {% for review in show.reviews %}
        <div class="card my-2">
            <div class="card-header">
                Critique écrite par : {{review.username}}
            </div>
            <div class="card-body">
                <blockquote class="blockquote mb-0">
                    <p> {{review.content}}</p>
                    <p> Note : </p>
                    {% include "fragments/_stars.html.twig" with {rating : review.rating} %}
                </p>
                <footer class="blockquote-footer">
                    {{review.watchedAt|date("d/m/Y")}}
                </footer>
            </blockquote>
        </div>
    {% endfor %}
</section>
```

Dans ce fichier, nous avons donc du HTML et une dynamisation en PHP avec le langage de template Twig. Dans mon code, l'entité Show (films ou séries) est liée à l'entité Review (commentaires) par une relation **One-To-Many**, c'est à dire qu'un film peut avoir plusieurs commentaires, mais un commentaire n'est relié qu'à un(e) film/série.

C'est grâce à cette relation que l'affichage conditionnel du titre `<h2>` des commentaires fonctionne.

```
● ● ●

    {% if show.reviews is not empty %}
        <h2 class="my-3">Les reviews</h2>
    {% else %}
        <h2 class="my-3">Pas encore de reviews</h2>
    {% endif %}
```



# DOSSIER PROFESSIONNEL (DP)

MINISTÈRE CHARGÉ  
DE L'EMPLOI

Ici, dans l'extrait de code précédent, je pose une condition grâce à la syntaxe Twig :

- S'il existe des commentaires pour ce film ou cette série, je peux afficher un titre <h2> spécifique :

The screenshot shows a movie review page for "La La Land". The page includes basic movie details like title, length, and genres. A large green rectangular area covers the main content area. Below it, a section titled "Les reviews" contains a single comment from a user named Pierre. The comment text is: "Superbe film hommage à toutes les plus grandes comédies musicales. De la magie, de la poésie, de la musique, bref, un moment inoubliable !". The rating for this comment is 5 stars.

- S'il n'y en a pas, on laissera le <h2> par défaut :

The screenshot shows the same movie review page for "La La Land" as the previous one. However, the "Les reviews" section is now empty, displaying the placeholder text "Pas encore de reviews".

Ensuite, quand il y a des commentaires, on boucle sur les commentaires existants avec `{% for review in show.reviews %}` pour afficher les différents commentaires.

Et c'est ici que la question de la modération des commentaires se pose : que faire en cas de spams ou de commentaires insultants par exemple ? Je ne l'ai pas mis en place lors de ce projet effectué au cours de ma formation, mais ma méthodologie serait la suivante.

Je mettrai en place un **CRUD** (Create, Read, Update, Delete) dans le backoffice du site O'flix pour que les administrateurs du site puissent avoir un moyen de supprimer les commentaires inutiles comme les spams ou les insultes gratuites. Par la suite, on pourrait même imaginer la mise en place de filtres sur certains mots ou sur un dictionnaire de mots reconnus comme étant des insultes afin d'émettre une notification par mail aux administrateurs dès qu'un commentaire de ce genre est posté.

## 2. Précisez les moyens utilisés :

Pour réaliser ce projet du cursus O'clock, j'ai utilisé l'éditeur de code Visual Studio Code.

Pour tester l'affichage de la page web, j'ai utilisé le navigateur web Google Chrome et notamment ses outils d'aide au développement pour faciliter l'intégration et le débogage de mon code.

J'ai également utilisé plusieurs documentation en ligne :

- Documentation de Symfony : <https://symfony.com/doc/current/index.html>
- Documentation du langage de template Twig : <https://twig.symfony.com/doc/3.x/>
- Documentation de Bootstrap pour la mise en page :  
<https://getbootstrap.com/docs/5.3/getting-started/introduction/>

## 3. Avec qui avez-vous travaillé ?

J'ai travaillé seul sur ce projet réalisé lors de mon cursus de formation chez O'clock.

## 4. Contexte

Nom de l'entreprise, organisme ou association ▶ **Ecole O'clock**

Chantier, atelier, service ▶ **Projet réalisé au cours de la formation O'clock.**

Période d'exercice ▶ Du : **30/10/2023** au : **22/04/2024**

## 5. Informations complémentaires (facultatif)



# DOSSIER PROFESSIONNEL (DP)

## Activité-type 2

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

**CP 8 - Documenter le déploiement d'une application dynamique web ou web mobile**

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

Une fois le projet terminé, je peux l'utiliser en local, mais ce que je veux surtout, c'est le partager au monde entier. Il convient alors de le déployer en ligne sur un serveur. Pour cette Compétence Professionnelle, je vais reprendre l'exemple du projet O'flix que j'ai utilisé dans la CP 5 et dans la CP 7. Pour rappel, O'flix est une base de données de films et séries semblable à un site comme [IMDB](#) :

- les utilisateurs peuvent se connecter et mettre des films/séries en favoris, noter et laisser un commentaire sur un film ou une série,
- les administrateurs ont accès à un backoffice qui permet de créer un nouveau film ou une nouvelle série.

Une fois le projet terminé, comment le déployer sur un serveur pour qu'il soit accessible à tout le monde ? C'est ce que nous allons voir maintenant.

Pour que mon site O'flix soit consultable en ligne, je vais déployer mon projet sur un serveur AWS Ubuntu, un serveur d'Amazon Web Service qui tourne sur le système d'environnement Linux. De base, j'installe de quoi faire tourner un site en PHP sur le serveur AWS Ubuntu :

- Un **serveur http** : Apache, qui permet une communication avec le navigateur via le protocole http(s).
- Un **système de gestion de base de données** : MariaDB, qui me permet de créer mes bases de données pour mes différents projets. Je gère ensuite mes bases de données via l'interface graphique Adminer.
- Un **langage de programmation interprété**, PHP, pour produire des pages web de façon dynamique. Il est couplé au serveur Apache pour assurer la communication avec le client web.

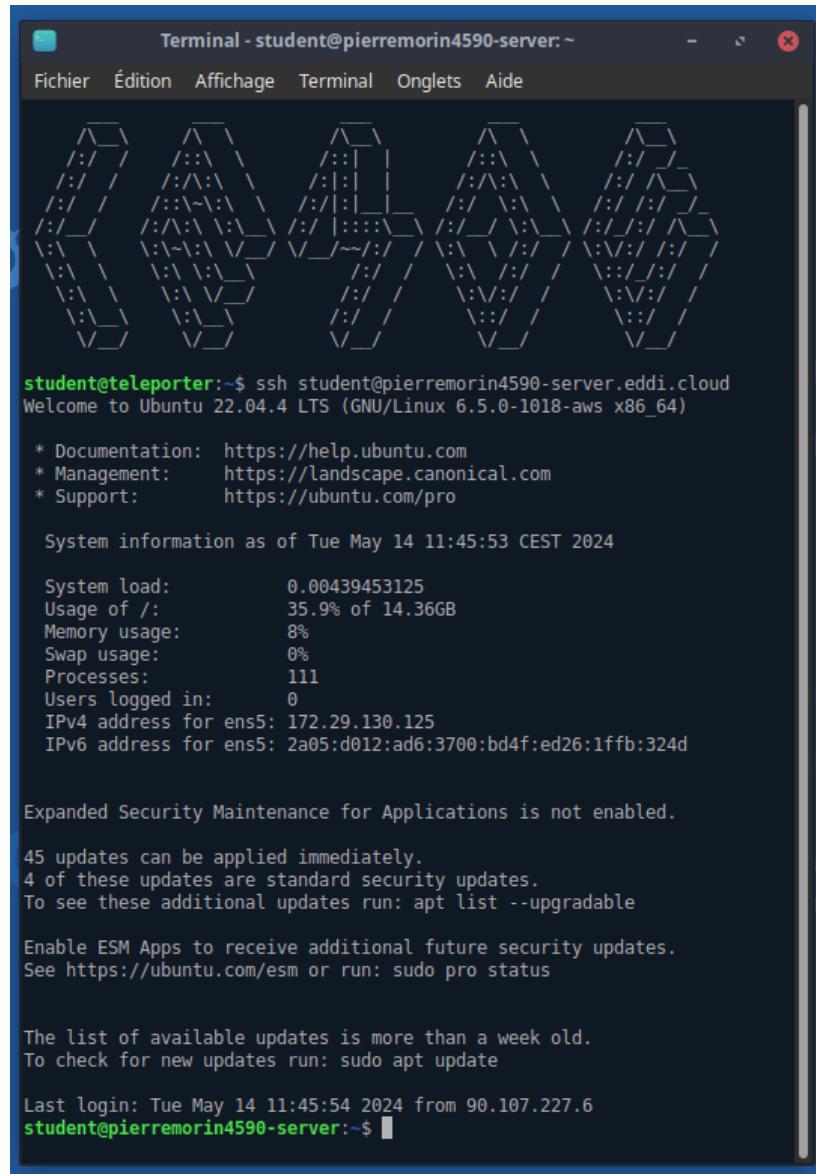
À noter, à ce stade, j'ai installé ce qu'on appelle un serveur web LAMP (Linux, Apache, MySQL, PHP) fonctionnel.

## **Etape 1 :** Effectuer quelques ajustements sur le serveur qui va accueillir le projet

Depuis la machine virtuelle que j'utilise pour coder (voir CP 1), j'ouvre le terminal pour me connecter à mon serveur AWS Ubuntu avec la commande suivante :

```
ssh student@pierremorin4590-server.eddi.cloud
```

Cette commande me permet de me connecter à un serveur distant via le protocole SSH. SSH, ou Secure Shell, est un protocole de communication sécurisé qui permet d'établir une connexion sécurisée avec un ordinateur distant sur un réseau.



The screenshot shows a terminal window titled "Terminal - student@pierremorin4590-server:~". The window has a menu bar with "Fichier", "Édition", "Affichage", "Terminal", "Onglets", and "Aide". Below the menu is a decorative ASCII art logo consisting of various symbols like slashes and dots forming a stylized shape. The terminal then displays the following text:

```
student@teleporter:~$ ssh student@pierremorin4590-server.eddi.cloud
Welcome to Ubuntu 22.04.4 LTS (GNU/Linux 6.5.0-1018-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/pro

 System information as of Tue May 14 11:45:53 CEST 2024

 System load:          0.000439453125
 Usage of /:            35.9% of 14.36GB
 Memory usage:         8%
 Swap usage:           0%
 Processes:             111
 Users logged in:      0
 IPv4 address for ens5: 172.29.130.125
 IPv6 address for ens5: 2a05:d012:ad6:3700:bd4f:ed26:1ffb:324d

 Expanded Security Maintenance for Applications is not enabled.

 45 updates can be applied immediately.
 4 of these updates are standard security updates.
 To see these additional updates run: apt list --upgradable

 Enable ESM Apps to receive additional future security updates.
 See https://ubuntu.com/esm or run: sudo pro status

 The list of available updates is more than a week old.
 To check for new updates run: sudo apt update

 Last login: Tue May 14 11:45:54 2024 from 90.107.227.6
student@pierremorin4590-server:~$
```

Une fois la commande effectuée, on voit bien que mon terminal peut désormais effectuer des actions sur une machine distante puisque le prompt de commande est passé de **student@teleporter** à **student@pierremorin4590-server**, je suis donc bien connecté à mon serveur.

Une fois connecté, nous pouvons effectuer nos ajustements.



# DOSSIER PROFESSIONNEL (DP)

Tout d'abord, j'active le module de réécriture Rewrite pour Apache avec la commande :

```
sudo a2enmod rewrite
```

Cela va me permettre de modifier les URL demandées par le client avant qu'elles ne soient traitées par le serveur. Je redémarre ensuite mon serveur pour activer la prise en compte de la réécriture avec la commande :

```
sudo service apache2 restart
```

J'ai ensuite installé quelques modules supplémentaires pour PHP 7.4, qui vont être utile à Symfony, avec la commande :

```
sudo apt install php7.4-curl php7.4-gd php7.4-zip
```

**php7.4-curl** → Ce module fournit une interface pour travailler avec des requêtes et des transferts de données via des URL.

**php7.4-gd** → Ce module est une bibliothèque pour la création et la manipulation d'images en PHP.

**php7.4-zip** → Ce module permet la manipulation des fichiers ZIP en PHP.

Ces trois modules sont souvent utilisés en développement web PHP pour des tâches comme l'interface avec une API (via CURL), l'utilisation des images et la gestion des fichiers compressés.

Une fois ces modules installés, j'installe également des librairies qui seront utiles à Composer avec la commande :

```
sudo apt install zip unzip
```

Cette commande installe les utilitaires de ligne de commande zip et unzip qui serviront à manipuler les archives compressées. J'installe ces librairies car Composer les utilise souvent pour extraire les dépendances lors de l'installation des packages PHP. Ce sont donc des outils importants pour le bon fonctionnement de Composer et de Symfony.

Je peux donc ensuite installer Composer avec les commandes :

```
php -r "copy('https://getcomposer.org/installer', '/tmp/composer-setup.php');"  
sudo php /tmp/composer-setup.php --install-dir=/usr/local/bin --filename=composer
```

La première commande permet de télécharger l'installateur Composeur depuis [le site de Composer](https://getcomposer.org/installer). La ligne de commande indique que cet installateur est téléchargé et enregistré dans le répertoire

temporaire **/tmp** et **composer-setup.php** est le nom sous lequel cet installateur est enregistré dans le fichier temporaire.

La deuxième ligne de commande exécute l'installateur de Composer téléchargé précédemment. L'exécutable est placé dans le répertoire **/usr/local/bin** qui est un emplacement souvent utilisé pour les exécutables système.

Une fois Composer installé, on vérifie que Composer est bien installé avec la commande :

```
composer --version
```

Si Composer est bien installé, le terminal devrait afficher la version de Composer :

```
student@pierremorin4590-server:~$ composer --version
Composer version 2.7.2 2024-03-11 17:12:18
PHP version 8.2.16 (/usr/bin/php8.2)
Run the "diagnose" command to get more detailed diagnostics output.
```

**Etape 2 :** Mise en production d'un projet Symfony existant

### **Gestion des droits Apache et user AWS Ubuntu**

Sur le serveur, pour être en mesure de cloner et écrire dans le répertoire **/var/www** je dois configurer les droits nécessaires aux utilisateurs de la machine distante. Pour cela, j'utilise les commandes suivantes :

```
sudo chown -R ubuntu.www-data /var/www
sudo chmod -R ug+rwx /var/www
```

Ces commandes assurent que l'utilisateur ubuntu a le contrôle total sur les fichiers dans le répertoire **/var/www** et que le groupe **www-data**, qui est généralement utilisé par le serveur web, dispose également des permissions pour accéder et manipuler les fichiers de ce répertoire.

### **Rapatrier le projet**

Pour rapatrier le projet, je me connecte à GitHub et je copie le lien SSH (configuration en amont avec la commande **ssh-keygen** : la clé privée est stockée sur le serveur et la clé publique sur GitHub) du projet.

Je clone le projet dans le répertoire **/var/www/html** avec la commande :

```
git clone git@github.com:O-clock-Jelly/symfo-oflix-PierreMorin4590.git
```

### **Installer le projet**

Une fois le projet cloné, je rentre dans le répertoire du projet avec la commande :

```
cd symfo-oflix-PierreMorin4590
```

Je lance la commande **composer install** pour télécharger et installer toutes les dépendances définies dans le fichier **composer.json** du projet.



MINISTÈRE CHARGÉ  
DE L'EMPLOI

# DOSSIER PROFESSIONNEL (DP)

Je crée un fichier **.env.local** à la racine du projet et je le modifie pour y noter mes variables d'environnement comme par exemple l'URL de ma base de données, la clé API OMDB, l'URL pour la connexion SMTP aux serveurs de MailJet, etc...

Pour ce projet, j'ai créé des fixtures, j'effectue donc les commandes suivantes pour créer la base de donnée et y envoyer celles-ci :

```
php bin/console cache:clear
php bin/console doctrine:database:create
php bin/console doctrine:migration:migrate
php bin/console doctrine:fixtures:load
```

Ensuite, nous pouvons à nouveau modifier le fichier **.env.local** pour remplacer **APP\_ENV=dev** en **APP\_ENV=prod**, cela va permettre à notre site d'utiliser les configurations disponibles dans le répertoire **config/packages/prod**. Par exemple, la debug toolbar Symfony ne sera plus affichée.

Généralement, les fixtures ne sont pas utilisées lors de la mise en production d'un projet puisque des données sont généralement déjà présentes. Ici, nous les utilisons pour avoir des données à afficher sur le site O'flix.

## .htaccess et VirtualHost

Dernière étape pour s'assurer que tout fonctionnera bien : j'installe le pack Apache pour Symfony avec la commande : **composer require apache-pack**

Cela va permettre la création du fichier **.htaccess** dans le répertoire **public** du projet. Ce fichier permet la prise en compte des réécritures grâce au **mod\_rewrite** précédemment activé.

J'active ensuite la réécriture pour mon site dans **/var/www/html** en effectuant les actions suivantes :

1. **cd /etc/apache2/sites-available/** (se rendre dans le fichier à configurer)
2. **sudo nano 000-default.conf** (ouvre un fichier de configuration)

3. Puis ajout du bloc de code à la fin du bloc **VirtualHost \*:80** (code encadré d'un rectangle bleu dans la capture ci-après



```
<VirtualHost *:80>

# ...

DocumentRoot /var/www/html

<Directory /var/www/html>
    AllowOverride All
    Order Allow,Deny
    Allow from All
</Directory>

</VirtualHost>
```

4. Enfin on redémarre Apache, pour prendre en compte les modifications, avec la commande :

```
sudo service apache2 restart
```

Les instructions précédentes permettent donc de configurer Apache pour prendre en compte les fichiers **.htaccess** et les réécritures d'URL dans notre projet Symfony.

Une fois tout cela effectué, notre projet est déployé et accessible en ligne. Il convient alors de se rendre sur l'URL de notre serveur pour vérifier que le site s'affiche bien.

## 2. Précisez les moyens utilisés :

Pour mettre en ligne le projet O'flix, j'ai utilisé :

- le serveur AWS Ubuntu mis à disposition par O'clock pour héberger nos projets de formation,
- GitHub pour stocker le repository de mon projet,
- le terminal de ma machine virtuelle Linux pour configurer mon serveur et y installer mon projet,
- le navigateur Google Chrome afin de m'assurer que mon projet était bien accessible en ligne et fonctionnel.

## 3. Avec qui avez-vous travaillé ?

J'ai travaillé seul sur ce projet réalisé lors de mon cursus de formation chez O'clock.



Liberté • Égalité • Fraternité  
RÉPUBLIQUE FRANÇAISE

MINISTÈRE CHARGÉ  
DE L'EMPLOI

# DOSSIER PROFESSIONNEL (DP)

## 4. Contexte

Nom de l'entreprise, organisme ou association ▶ **Ecole O'clock**

Chantier, atelier, service ▶ **Projet réalisé au cours de la formation O'clock.**

Période d'exercice ▶ Du : **30/10/2023** au : **22/04/2024**

## 5. Informations complémentaires (facultatif)

## **Titres, diplômes, CQP, attestations de formation**

*(facultatif)*

<b>Intitulé</b>	<b>Autorité ou organisme</b>	<b>Date</b>
Cliquez ici.	Cliquez ici pour taper du texte.	Cliquez ici pour sélectionner une date.



MINISTÈRE CHARGÉ  
DE L'EMPLOI

# DOSSIER PROFESSIONNEL (DP)

## Déclaration sur l'honneur

Je soussigné Pierre Morin,

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

Fait à Bourges le mardi 14 mai 2024

pour faire valoir ce que de droit.

Signature :

## Documents illustrant la pratique professionnelle

(*facultatif*)

Intitulé
Cliquez ici pour taper du texte.



MINISTÈRE CHARGÉ  
DE L'EMPLOI

# DOSSIER PROFESSIONNEL (DP)

## ANNEXES

*(Si le RC le prévoit)*