Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE

MINISTÈRE CHARGÉ
DE L'EMPLOI

DOSSIER PROFESSIONNEL (DP)

Nom de naissance ▸ LIAUTAUD
Nom d'usage ▸ LIAUTAUD
Prénom ▸ Elise
Adresse ▸ 25 impasse de l'auberte 83210 LA FARLEDE

Titre professionnel visé

Développement Web Web Mobile

MODALITÉ D'ACCÈS :

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

Présentation du dossier

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

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

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

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

Pour prendre sa décision, le jury dispose :

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

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

Ce dossier comporte :

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

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

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

Sommaire

Exemples de pratique professionnelle

EXEMPLES DE PRATIQUE PROFESSIONNELLE

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

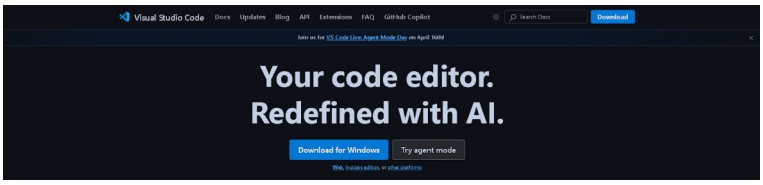
Exemple n°1 ▸ ***Installer et configurer son environnement de travail en fonction du projet web ou web mobile***

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

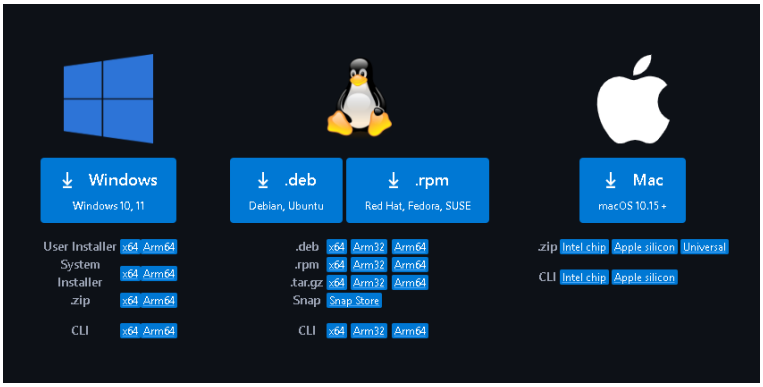
1. Installation et configuration de Visual Studio Code

La première étape primordiale dans la mise en place d'un environnement de développement est l'installation d'un IDE (Integrated Development Environment). Il en existe plusieurs, mais Visual Studio Code s'impose comme un choix idéal grâce à sa gratuité, sa fluidité, la richesse de ses extensions, son caractère open source, son système d'auto-complétion et sa grande communauté d'utilisateurs.

Pour le télécharger, je me rends sur le site officiel de VS Code : https://code.visualstudio.com, je clique sur le bouton "**Download"** et je choisis l'installation correspondant à mon système d'exploitation (dans mon cas : Windows).

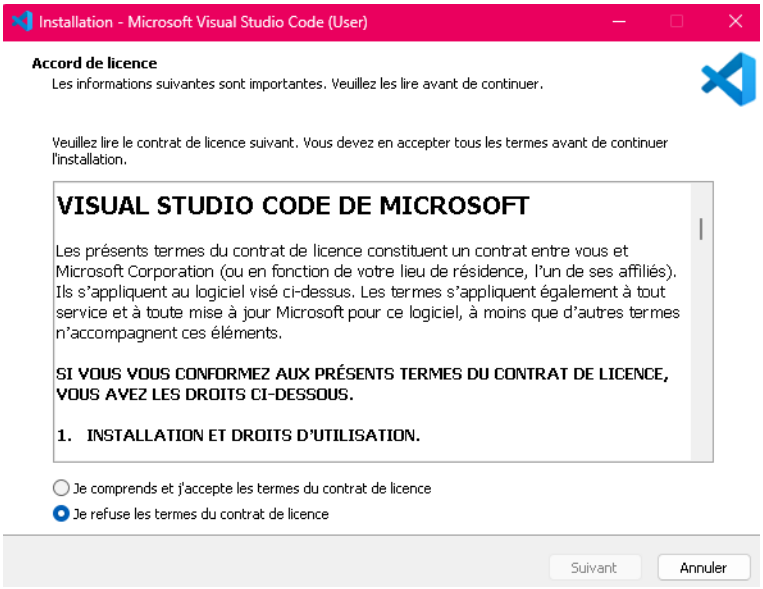


Bouton "Download"



Choix des options d'installations

Une fois le fichier d'installation téléchargé, je l'ouvre. L'assistant d'installation s'affiche et je suis les étapes classiques : accepter les termes du contrat de licence, cliquer sur "**Suivant"** jusqu'à la dernière étape, puis lancer l'installation en cliquant sur "**Installer"**



Page d'installation de VS Code

Une fois l'installation terminée, je peux lancer et utiliser Visual Studio Code. Pour optimiser mon espace de travail, j'installe ensuite plusieurs extensions indispensables.

Installation d'extensions

Pour installer une extension, j'ouvre VS Code , je clique sur l'icône "**Extensions"** dans la barre latérale gauche, puis je saisis le nom de l'extension dans la barre de recherche.

- **GitLens** : permet de visualiser et de mieux gérer l'historique Git directement dans l'éditeur.
- **Prettier** : formateur de code qui assure une mise en page cohérente.
- **ESLint** : outil d'analyse statique pour détecter et corriger des erreurs dans le code JavaScript.
    - Installation complémentaire dans le projet : commandes `npm install eslint --save-dev`et `npx eslint --init` dans le terminal
- **PHP Intelephense** : extension pour améliorer l'autocomplétion et l'analyse de code PHP.

Je tape leurs noms dans la barre de recherche d'extensions, et plusieurs choix d'extensions vont apparaître. Il est important de trouver l'extension qui correspond à ma recherche. Il suffit ensuite de cliquer sur le bouton "Install", et l'installation est terminée.

Avec VS Code et ces extensions configurées, mon environnement de développement est prêt à être utilisé.

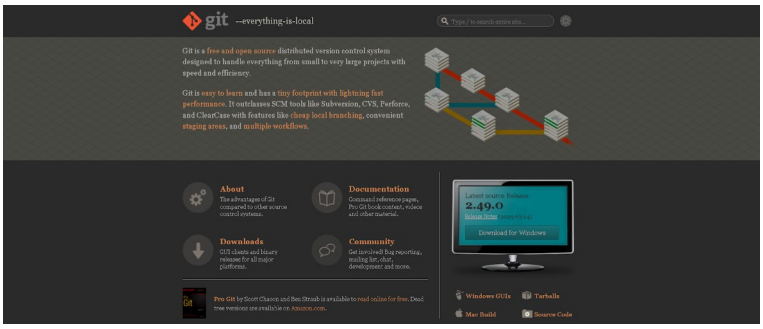
2. Configuration de Git

Pour garder un historique complet de mes projets et gérer les différentes versions de mon code, il est essentiel d'utiliser Git et GitHub.

Installation de Git

Je commence par télécharger **Git SCM** depuis le site officiel : https://git-scm.com.

- Je clique sur **Download** et choisis l'installateur adapté à mon système d'exploitation.
- Je sélectionne ensuite la dernière version disponible, puis lance le fichier téléchargé.



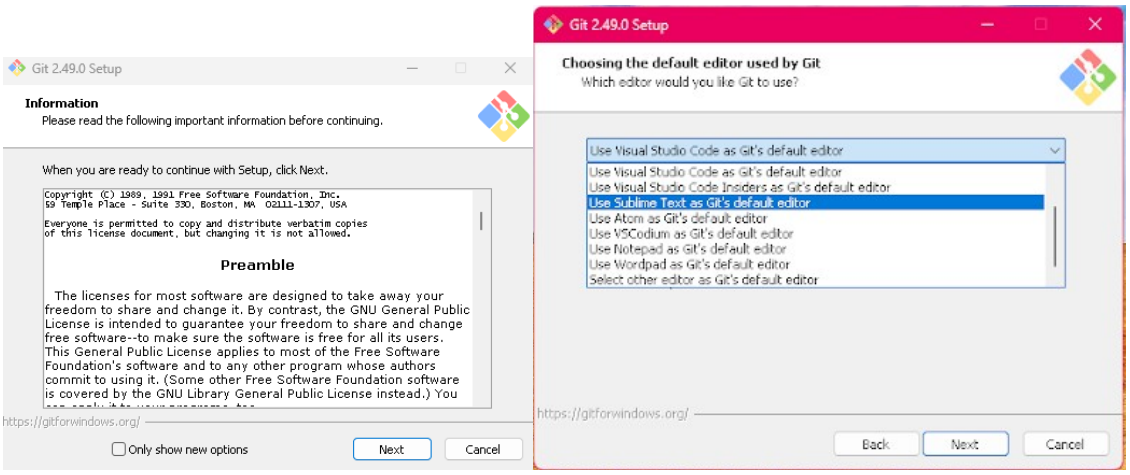
Bouton "Download"

Choix des options d'installations

Choix de la dernière version à installer

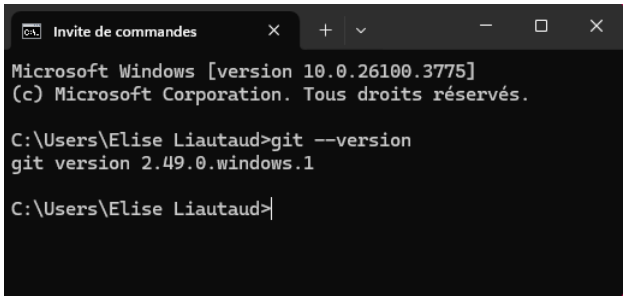
Lors de l'installation, il suffit de suivre l'assistant :

- Cliquer sur **Next** jusqu'à l'étape où l'on me demande de choisir l'éditeur par défaut.
- Sélectionner "**Use Visual Studio Code as Git's default editor"**.
- Continuer avec **Next** puis lancer l'installation avec **Install**.



Page d'installation de Git. Panneau déroulant à modifier.

Une fois l'installation terminée, j'ouvre l'Invite de commandes (cmd dans la barre de recherche Windows) et je vérifie que Git est bien installé avec `git --version`. Un message du type git version 2.49.0.windows.1 doit s'afficher (la version dépend de la date d'installation).



Configuration de l'identité utilisateur

Pour que Git identifie correctement mes commits, je dois renseigner mon nom et mon adresse e-mail :

- `git config --global user.name "*Votre Nom*"`
- `git config --global user.email "*votre@email.com*"`

```
C:\Users\Elise Liautaud>git config --global user.name "Elise L

C:\Users\Elise Liautaud>git config --global user.email "elli3183@gmail.com"
```

Si l'éditeur par défaut n'a pas été configuré correctement, je peux le forcer avec `git config --global core.editor "code –wait"`

Initialisation d'un dépôt Git

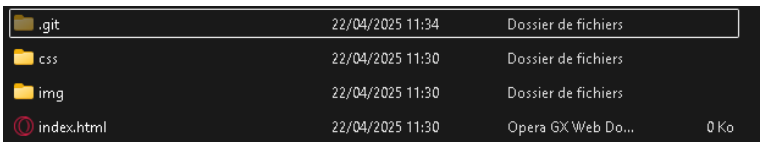
Je me place dans le dossier du projet (par exemple : C:\wamp64\www\2025\rapport) avec `cd C:\wamp64\www\2025\rapport`

```
C:\Users\Elise Liautaud> cd C:\wamp64\www\2025\rapport

C:\wamp64\www\2025\rapport>
```

Puis j'initialise le dépôt avec `git init`

```
C:\wamp64\www\2025\rapport>git init
Initialized empty Git repository in C:/wamp64/www/2025/rapport/.git/

C:\wamp64\www\2025\rapport>
```

Un dossier masqué .git est alors créé à la racine du projet (visible en activant l'affichage des éléments masqués dans l'explorateur de fichiers).



Création du fichier .gitignore

Pour éviter de versionner certains fichiers ou dossiers (dépendances, configurations locales, logs, builds, etc.), je crée un fichier **.gitignore** à la racine du projet.

fichier .gitignore à la racine

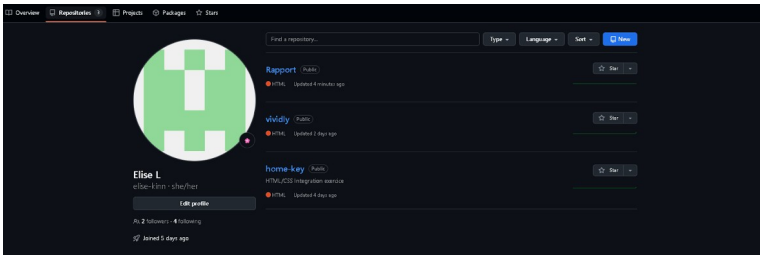
```
1  bin
2  # Ceci est un commentaire
```

À l'intérieur de .gitignore → dossier bin ignoré

Envoi du projet sur GitHub

Pour partager mon projet :

1. Je crée un nouveau **repository** depuis mon compte GitHub (*Profil → Repository → New*).



2. Je copie l'URL du repository généré.
3. Dans l'invite de commandes, je me replace dans mon projet et saisis :
    - git add *
    - git commit -m "*MESSAGE*"
    - git branch -m main
    - git remote add origin *URL_DU_REPO*
    - git push -u origin main

Après actualisation de la page GitHub, mon projet est visible dans le repository : https://github.com/elise-kinn/Rapport

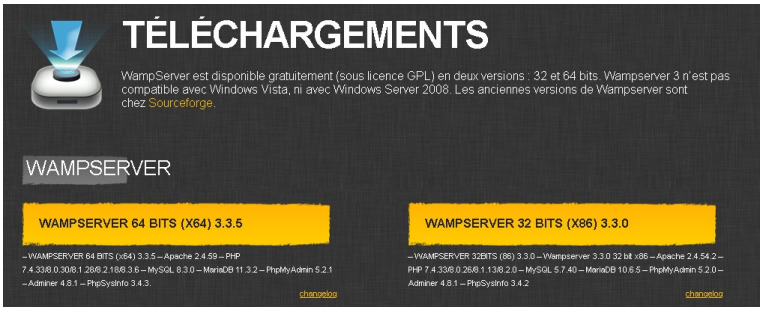
3. Mise en place d'un serveur local

Pour tester un projet en PHP et gérer une base de données MySQL en local, il est nécessaire d'installer un serveur local. Plusieurs solutions existent (XAMPP, WAMP, MAMP), mais j'ai choisi **WAMP**.

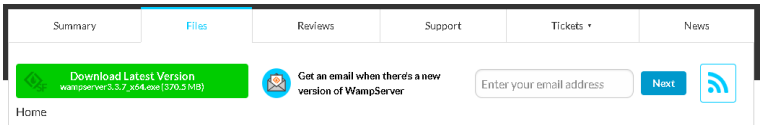
Téléchargement et installation de WAMP

1. Je me rends sur le site officiel : https://www.wampserver.com.
2. Je clique sur **Télécharger**, puis sur **WampServer 64 bits (x64) 3.3.5**.
3. Le site propose un formulaire, mais je peux passer directement au téléchargement en cliquant sur "*passer au téléchargement direct*".
4. Je clique ensuite sur le bouton vert "**Download Latest Version"** et j'enregistre le fichier d'installation.

"Télécharger"



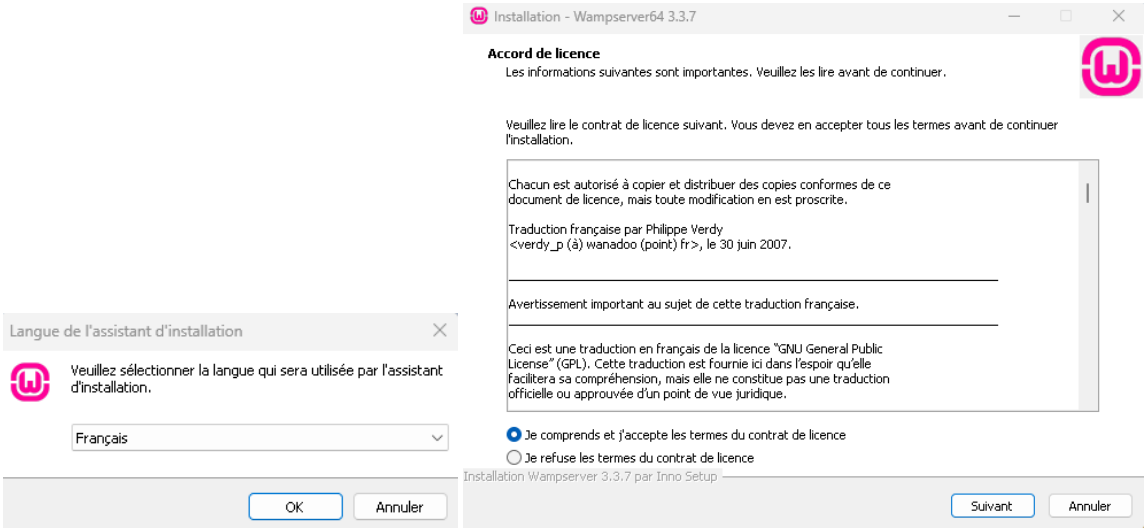
""WampServer 64 bits (x64) 3.3.5"



"Download Latest Version"

Une fois téléchargé, je lance le fichier d'installation :

- Choix de la langue.
- Acceptation des termes du contrat de licence.

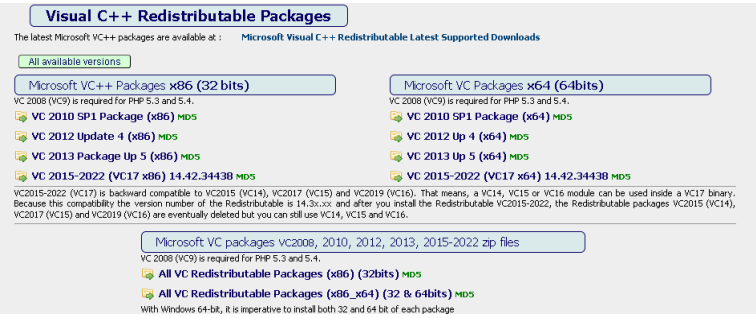


Vérification des dépendances Visual C++

Avant l'installation complète, WAMP vérifie la présence des paquetages Visual C++ nécessaires à son bon fonctionnement.

- Je clique sur le lien du paragraphe "*Paquetage V++*" pour télécharger check_vcredist.exe.
- Je lance l'outil, qui me signale si certains paquetages sont manquants.
- Si nécessaire, je me rends sur https://wampserver.aviatechno.net pour télécharger et installer tous les paquetages requis.
- Je relance l'outil afin de confirmer que tout est bien installé.

Une fois cette étape validée, je peux reprendre l'installation de WAMP et cliquer sur **Next** jusqu'à la fin du processus.

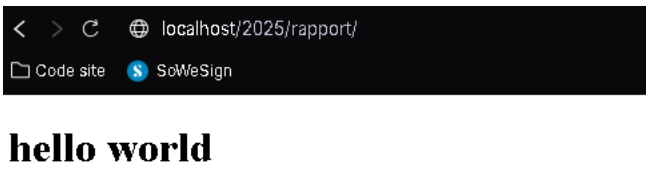


https://wampserver.aviatechno.net pour installer les paquetages.

Utilisation de WAMP

Pour démarrer **Apache** et **MySQL**, j'ouvre **Wampserver64**. Tous les services se lancent automatiquement.

- Je place ensuite mon projet dans le dossier www de l'installation WAMP (exemple : C: \wamp64\www\2025\rapport).
- Enfin, j'accède à mon projet depuis le navigateur en tapant http://localhost/2025/rapport/. Le projet s'affiche alors sur le serveur local.



3. Préparation d'un projet

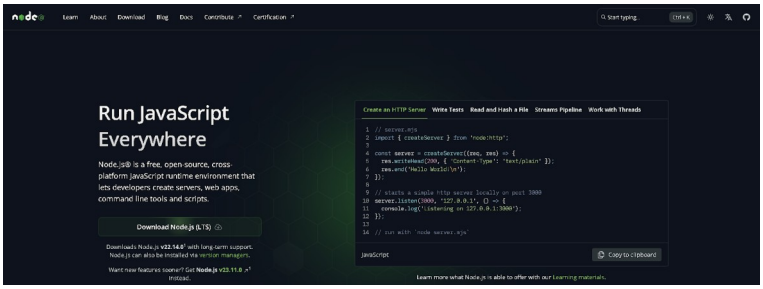
Pour mettre en place un projet côté serveur, deux environnements principaux s'offrent à moi : **Node.js** (JavaScript côté serveur) et **PHP** (via WAMP).

Installation de Node.js

Node.js permet d'exécuter du code JavaScript côté serveur. Pour l'installer :

1. Je me rends sur le site officiel : https://nodejs.org/en.
2. Je clique sur "**Download Node.js (LTS)**".

3. Je télécharge et lance le fichier d'installation, puis je suis les étapes proposées.



Une fois l'installation terminée, je vérifie que Node.js et son gestionnaire de paquets (npm) sont bien installés en ouvrant l'invite de commande et en tapant :

- node -v
- npm -v

Ces commandes doivent retourner les numéros de version installés :

```
C:\Users\Elise Liautaud>node -v
v22.14.0

C:\Users\Elise Liautaud>npm -v
10.9.2
```

Création d'un projet Node.js

Je crée un projet Node.js en utilisant les commandes suivantes :

- mkdir mon-projet-node
- cd mon-projet-node
- npm init -y
- npm install express

Ensuite, je crée un fichier **index.js** avec le code suivant pour démarrer un serveur HTTP simple avec **Express** :

```
1  const express = require('express');
2  const app = express();
3  app.get('/', (req, res) => res.send('Hello World!'));
4  app.listen(3000, () => console.log('Server running on port 3000'));
```

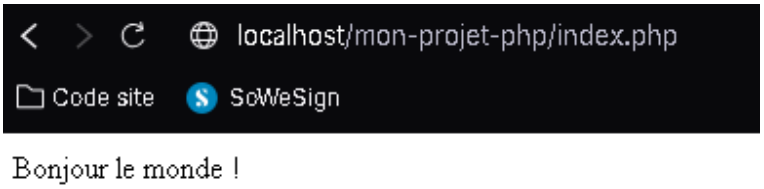
En lançant la commande node index.js, le serveur démarre, et je peux y accéder dans mon navigateur à l'adresse http://localhost:3000.

Création d'un projet PHP

Pour tester un projet PHP avec WAMP, je crée un dossier mon-projet-php dans C:\wamp64\www\.
À l'intérieur, j'ajoute un fichier **index.php** avec le code suivant :

```php
<?php
echo "Bonjour le monde !";
?>
```

Ensuite, j'ouvre mon navigateur et j'accède à l'adresse suivante : http://localhost/mon-projet-php/. Le message s'affiche, ce qui confirme que le serveur local est opérationnel.



2. Précisez les moyens utilisés :

Google, VS Code, Ordinateur

3. Avec qui avez-vous travaillé ?

Seule

4. Contexte

DOSSIER PROFESSIONNEL (DP)

Nom de l'entreprise, organisme ou association ▸	**Formation La Plateforme**
Chantier, atelier, service ▸	Atelier
Période d'exercice ▸	**Du** 22/04/2025 **au** 22/04/2025

Exemple n°2 ▸ ***Réaliser des interfaces utilisateur statiques web ou web mobile***

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

Description du projet : Site vitrine statique présentant profil professionnel, compétences, projets et coordonnées. Le site doit être conçu uniquement avec HTML5 et CSS3, sans frameworks ni bibliothèques externes (pas de Bootstrap ni JavaScript). Maquette Figma modèle à disposition.

1. Création du projet

La création d'un nouveau projet nécessite des étapes préliminaires importantes afin de travailler efficacement.

La première étape consiste à créer un dossier spécifique au projet, qui contiendra l'ensemble des documents nécessaires. J'ai donc créé un dossier nommé "**Portfolio ORAL"**. Le nom d'un dossier doit être clair et unique afin d'éviter toute confusion avec d'autres projets.

Dans ce dossier, j'ai organisé mon espace de travail en ajoutant :

- Les quatre fichiers HTML attendus (**index.html**, **contact.html**, **project.html** et **about.html**) ;
- Un dossier **CSS** contenant le fichier **style.css** ;
- Un dossier **img** destiné à stocker l'ensemble des images utilisées dans le portfolio ;
- Un dossier pour chacun des projets présentés (ici deux), afin de pouvoir y accéder directement sans avoir besoin de les héberger ailleurs.

Enfin, j'ai créé un **repository GitHub** et relié celui-ci à mon dossier local, afin de sauvegarder régulièrement mon avancement de manière sécurisée et d'éviter toute perte de données en cas de problème.

2. Initialisation HTML / CSS

Pour démarrer mon travail, j'ai pris l'habitude d'initialiser mes fichiers HTML et CSS de la même manière à

chaque projet, afin de gagner du temps et de faciliter mon avancée.

HTML

```html
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Porfolio de Elise : Accueil</title>

    <link rel="stylesheet" href="css/style.css">
    <script src="https://kit.fontawesome.com/f56ca831ee.js" crossorigin="anonymous"></script>
</head>
```

Tout d'abord, il est important de commencer le fichier HTML par des lignes précises :

- <!DOCTYPE html> indique au navigateur le langage utilisé
- <html lang="fr"> précise la langue du site, utile notamment pour les lecteurs d'écran qui se basent sur cette information

Dans la balise <head> :

- <meta charset="UTF-8"> définit l'encodage des caractères du document
- <meta name="viewport" content="width=device-width, initial-scale=1.0"> rend la page responsive sur tous les appareils
- <title> permet d'inscrire le titre de la page, affiché dans l'onglet du navigateur
- <link rel="stylesheet" href="css/style.css"> relie le fichier HTML au fichier CSS, en indiquant le chemin correspondant ;
- <script> permet d'intégrer des ressources externes, comme la galerie d'icônes gratuite **FontAwesome**. Après création d'un compte sur le site, il suffit de copier-coller la ligne proposée.

CSS

```css
/* Font Awesome link */
@import url('https://fonts.googleapis

/* ---------------------------------

*{
    margin: 0;
    list-style: none;
    text-decoration: none;
    scroll-behavior: smooth;
    transition: all 0.2s ease;
    color : var(--dark-pink);
    font-family: "Lato", sans-serif;
    object-fit: cover;
}
```

```css
:root{
    --pale-rose : #fff1fde4;
    --deep-pink : #5f1443;
    --dark-pink : #4c3a44;
    --pink-button : #70345e;
    --pink-span : #ddc4d1e4;
    --green-button : #5aa874;
    --green-link : #4b904b;
    --pink-nav : #e966a0cc;
    --pink-bandreau : #8a7482;
}
```

```css
/*Fonts*/

h1, .name, h2, h3{
    font-family: "Fraunces", serif;
    font-weight: 700;
}

h2{
    font-size: 41px;
    text-align: center;
    margin-bottom : 16px
}

h3{
    font-size: 29px;
    margin-bottom : 19px;
}
```

```css
#bars{ /*responsive nav*/
    display: none;
}

header>div, #about, footer>div, #projects>div, #profile>div{
    max-width: 1440px;
    margin : auto;
}
```

Pour mon fichier CSS, j'utilise également une structure d'initialisation systématique :

- @import url() permet d'importer des polices gratuites depuis **Google Fonts**, que j'ai sélectionnées pour le projet
- Le sélecteur universel * applique des règles à tous les éléments : suppression des marges par défaut, retrait des styles automatiques sur les listes, ajout de transitions globales pour fluidifier les effets de survol, définition de la police et de la couleur de texte, scroll fluide avec scroll-behavior: smooth et ajustement des images avec object-fit: cover
- Le sélecteur **:root** me permet de créer des variables, ici principalement pour les couleurs utilisées de manière répétée dans le projet
- Je normalise ensuite les styles de texte avec des sélecteurs spécifiques, en définissant la police, la taille, l'épaisseur et les marges si nécessaire ;
- Enfin, pour les sections, j'applique max-width: 1440px et margin: auto afin de centrer le contenu et de limiter la largeur maximale sur les grands écrans.

L'initialisation est désormais terminée. Je peux commencer à construire le contenu de ma page.

3. Header

Le **header** correspond au haut de ma page. Il contient mon **logo cliquable** (ici simplement mon nom, puisqu'il s'agit d'un portfolio) ainsi que la **barre de navigation**, qui permet de naviguer entre les quatre pages du site.

Pour la mise en page, j'utilise display: flex afin de positionner les éléments aux extrémités du header avec justify-content: space-between.

Logo

Le logo est entouré d'une balise <a>, qui renvoie vers **index.html**, conformément à la convention pour offrir une meilleure expérience utilisateur.

Barre de navigation

La barre de navigation est placée dans une balise <nav>, importante pour le référencement. À l'intérieur, j'utilise :

- une balise <ul> pour créer une liste non ordonnée
- quatre <li> correspondant aux différentes pages, chacun contenant une **ancre** <a> pour rendre les éléments cliquables
- une **vide** utilisée pour l'effet visuel lors du survol.

Par défaut, cette div a une certaine **hauteur** mais une **largeur nulle**, ce qui la rend invisible. Lors du survol, la largeur passe à 100 %, elle s'étend donc sur toute la largeur du <li> de manière fluide, donnant un effet d'apparition progressive depuis la gauche.

Pour indiquer la page active, j'ajoute la **classe** pink sur l'élément actif.

Adaptation mobile

Pour les petits écrans, la barre de navigation disparaît afin de laisser place à une icône de menu.

- J'attribue un **id** à la barre de navigation pour faciliter sa sélection en CSS ;
- J'insère également l'icône avec un **id** distinct.
- Pour l'instant, l'icône est masquée avec display: none.

```
<header>
    <div>
        <a href="index.html" class="name">Elise LIAUTAUD</a>
        <nav>
            <ul id="nav">
                <li><a href="index.html" class="pink">Accueil</a><div></div></li>
                <li><a href="about.html">À propos</a><div></div></li>
                <li><a href="projects.html">Projets</a><div></div></li>
                <li><a href="contact.html">Contact</a><div></div></li>
            </ul>
            <i class="fa-solid fa-bars" id="bars"></i>
        </nav>
    </div>
</header>
```

```
#bars{ /*responsive nav*/
    display: none;
}
```

```css
header{
    background-color: var(--pale-rose); /*

    > div{
        display: flex;
        justify-content: space-between;
        align-items: center;
        padding : 25px 15px;

        .name{
            font-size: 24px;
            color : var(--dark-pink);
        }

        nav ul{
            display: flex;
            gap : 15px;

            a{
                font-weight: 600;
            }
```

```css
            div{
                margin-top : 2px;
                height : 2px;
                width :0%; /*invisible*/
                background-color: var(--pink-button);
            }

            li:hover{
                a{
                    color: var(--green-button);
                }

                div{
                    width : 100%
                }
            }

            .pink{
                color : var(--pink-nav);
            }
        }
    }
}
```

4. Footer

Le **footer** correspond au bas de ma page. Il contient :

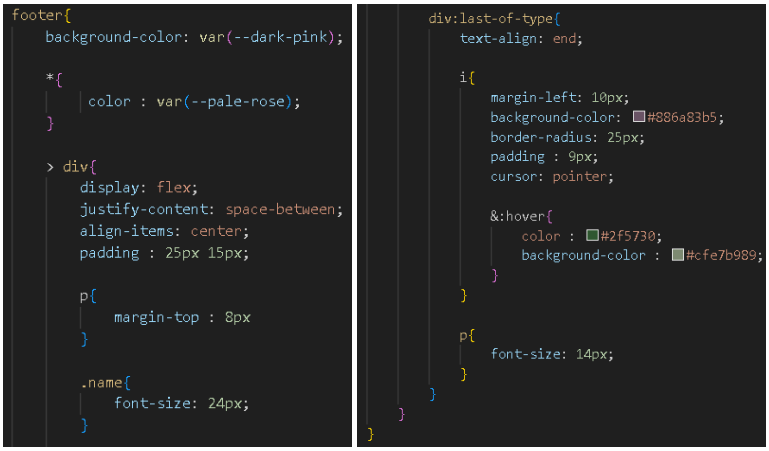
- mon **logo cliquable**, comme dans le header ;
- mon **métier**
- des **liens vers mes réseaux sociaux**
- les **droits d'auteur** du site

Pour la mise en page, j'utilise display: flex et justify-content: space-between, afin de disposer les deux horizontalement et de les espacer correctement.

Une regroupe l'ensemble des icônes. Cela facilite le centrage et l'organisation des icônes, notamment pour l'affichage **responsive**.

Pour les liens vers les réseaux sociaux, j'ai ajouté un **effet au survol**, qui modifie à la fois la couleur de l'icône et celle de son arrière-plan, afin de renforcer l'interactivité et la visibilité.

```html
<footer>
    <div>
        <div>
            <a href="index.html" class="name">Elise LIAUTAUD</a>
            <p>Développeuse Web Front-End & Designer UI/UX</p>
        </div>
        <div>
            <div id="icone-container"><!-- responsive -->
                <a href="https://github.com/elise-kinn" target="_blank"><i class="fa-brands fa-github"></i></a>
                <a href=""><i class="fa-brands fa-linkedin-in"></i></a>
            </div>

            <p>© 2025 Elise LIAUTAUD. Tous droits réservés.</p>
        </div>
    </div>
</footer>
```



```scss
footer{
    background-color: var(--dark-pink);

    *{
        color : var(--pale-rose);
    }

    > div{
        display: flex;
        justify-content: space-between;
        align-items: center;
        padding : 25px 15px;

        p{
            margin-top : 8px
        }

        .name{
            font-size: 24px;
        }

        div:last-of-type{
            text-align: end;

            i{
                margin-left: 10px;
                background-color: #886a83b5;
                border-radius: 25px;
                padding : 9px;
                cursor: pointer;

                &:hover{
                    color : #2f5730;
                    background-color : #cfe7b989;
                }
            }

            p{
                font-size: 14px;
            }
        }
    }
}
```

5. Section Profile

Une fois le **header** et le **footer** terminés, je peux commencer à construire le **contenu principal** de mon site, placé à l'intérieur de la balise <main>.

Pour la section **Profile** :

- J'utilise display: flex et justify-content: space-between pour organiser horizontalement les éléments
- L'<article> et la (contenant la photo) reçoivent flex: 1 pour répartir l'espace de manière égale.
- J'ajoute un ::after à la contenant ma photo de profil pour créer un élément décoratif supplémentaire.

Fond coloré et décoratif

Pour le fond de cette section :

- Je le crée via un ::before, afin de pouvoir appliquer des filtres ou autres effets visuels sans impacter les éléments à l'intérieur de la section
- content: "" est nécessaire pour afficher le ::before ;
- position: absolute permet de positionner le fond sur la section (le conteneur parent doit être en position: relative) ;
- Le visuel du fond est un motif de pois, défini par :
    - background-image pour la forme et les couleurs des pois ;
    - background-size pour la taille des pois ;
    - background-position pour leur positionnement ;

- background-color pour la couleur de fond.

```html
<section id="profile">
    <div>
        <article>
            <h1>Elise LIAUTAUD</h1>
            <p>Développeuse Web Front-End & Designer UI/UX</p>
            <p>Je crée des expériences web élégantes, intuitives et performantes avec une attention particulière aux détails.</p>
            <a href="projects.html" class="button green">Voir mes projets</a>
        </article>
        <div>
            <img src="img/profile-pic.jpg" alt="profile picture">
        </div>

    </div>
</section>
```

```css
#profile{
    position : relative;

    &:before{
        content :"";

        background-image:
            radial-gradient(#944594 4px, transparent 4px),
            radial-gradient(#7c657c 4px, transparent 4px);
        background-size: 73px 73px;
        background-position: 0 0, 36.5px 36.5px;
        background-color: rgb(89 4 67 / 45%);

        width: 100%;
        height: 100%;
        position: absolute;
        z-index: -1;
    }

    > div{
        display: flex;
        align-items: center;
        justify-content:space-between;
        padding : 200px 15px;
        gap : 100px;

        article, div{
            flex : 1 /*division even*/
        }

        *{
            color : var(--pale-rose);
        }
```

```css
        h1{
            font-size: 60px;
        }

        p:first-of-type{
            font-size: 24px;
            margin-top : 4px;
            margin-bottom : 30px;
        }

        p:last-of-type{
            margin-bottom : 33px;
            font-size: 18px;
        }

        img{
            border-radius: 185px;
            border: 4px dashed #534352;
            width: 50%;
            margin-left : calc(50% - 135px);
        }
```

```css
        > div{
            position: relative;

            &::after{
                position: absolute;
                content: "</>";
                font-size: 50px;
                font-weight: 500;
                background: #041f00;
                background: linear-gradient(
                    308deg, rgb(65 120 57) 0%,
                    rgb(205 220 203) 100%
                );
                color: #678f5c;
                border-radius: 50px;
                bottom: -20px;
                right: 170px;
                padding: 14px 18px 18px 18px;
                z-index: 2;
            }
        }
    }
}
```

6. Section About

La section **#about** se divise en deux parties distinctes : un <article> et une , organisés horizontalement et répartis équitablement dans l'espace grâce à display: flex et flex: 1.

Compétences

La partie présentant mes **compétences** utilise display: grid pour organiser les éléments sous forme de grille :

- grid-template-columns: 1fr 1fr 1fr crée deux lignes et trois colonnes
- À l'intérieur de chaque article, j'applique display: flex, flex-direction: column et align-items: center pour centrer les éléments verticalement et horizontalement.

Pour renforcer l'expérience utilisateur, chaque article possède un **effet au survol**, modifiant la

couleur et la taille des icônes et des titres de manière interactive.

```html
<section id="about">
    <h2>À propos de moi</h2>
    <div>
        <article>
            <h3>Mon parcours</h3>
            <p>
                Mon équivalent BAC+2 en Développement Web en poche,  je combine des compétences techniques solides avec une sensibilité esthétique pour créer des sites web et applications qui sont à la fois fonctionnels et visuellement attrayants.
            </p>
            <p>
                Après 3 mois en tant qu'auto-didacte et 9 mois au sein de la formation Développement Web - Web Mobile (La Plateforme), j'ai développé une expertise dans la création d'interfaces utilisateur intuitives et accessibles, en privilégiant toujours l'expérience utilisateur.
            </p>
            <p>
                Je suis passionnée par les technologies web modernes et je m'efforce constamment d'améliorer mes compétences pour rester à la pointe des dernières tendances et meilleures pratiques.
            </p>
            <a href="about.html" class="link">En savoir plus sur mon parcours <i class="fa-solid fa-arrow-right"></i></a>
        </article>

        <div>
            <h3>Mes compétences</h3>
            <div>
                <article>
                    <i class="fa-brands fa-html5"></i>
                    <p>HTML5</p>
                </article>

                <article>
                    <i class="fa-brands fa-css3"></i>
                    <p>CSS3</p>
                </article>

                <article>
                    <i class="fa-brands fa-js"></i>
                    <p>JavaScript</p>
                </article>

                <article>
                    <i class="fa-solid fa-code-branch"></i>
                    <p>Git</p>
                </article>

                <article>
                    <i class="fa-solid fa-object-group"></i>
                    <p>UI/UX Design</p>
                </article>

                <article>
                    <i class="fa-solid fa-mobile-screen-button"></i>
                    <p>Responsive</p>
                </article>
            </div>
        </div>
    </div>
</section>
```

```scss
#about{
    padding : 70px 15px;

    >div{
        display: flex;
        gap : 50px;
        align-items: center;
        margin-top : 30px;

        p{
            margin-bottom : 22px
        }

        >div, >article{
            flex : 1 /*division even*/
        }

        >div >div{
            display: grid;
            grid-template-columns: 1fr 1fr 1fr;
            row-gap : 23px;
            margin-top : 30px;

            article{
                display: flex;
                flex-direction: column;
                align-items: center;

                &:hover{
                    i{
                        color: var(--green-button);
                        transform: scale(1.05);
                    }

                    p{
                        color: var(--green-link);
                        transform: scale(1.05);
                    }
                }

                i{
                    font-size: 46px;
                    margin-bottom : 13px;
                    color: #c480ac;
                }
            }
        }
    }
}
```

7. Section Projets

La section **projects** présente deux de mes projets d'apprentissage HTML/CSS.

Chaque projet est entouré d'un <article>, placé dans une parente avec display: flex. À l'intérieur de

chaque article :

- Une **capture d'écran** du projet
- Un **titre**
- Une **description**
- Des **mots-clés** représentant les compétences utilisées
- Deux **liens**, vers le repository GitHub ou vers le projet lui-même

Mots-clés

Les mots-clés sont entourés de pour leur appliquer une couleur distincte. Pour permettre un retour à la ligne lorsque l'article rétrécit, ils sont regroupés dans une avec display: flex et flex-wrap: wrap. Chaque possède également un **effet au survol** pour renforcer l'interactivité.

Effets au survol des articles

Chaque article comporte plusieurs effets lors du survol :

- background-color change et un transform: rotate() applique une légère rotation ;
- l'image reçoit un filter() et un scale() pour un léger rétrécissement ;
- des éléments décoratifs contenus dans ::before et ::after apparaissent progressivement.
    - Par défaut, ces éléments ont opacity: 0 ;
    - Lors du survol, une animation **@keyframes** fait passer leur opacité à 1.
    - Cette approche permet une apparition fluide sans modifier la position des autres éléments autour.

```
<section id="projects">
    <div>
        <h2>Mes projets récents</h2>
        <p>Découvrez une sélection de mes travaux récents qui démontrent mes <br> compétences en développement web et design d'interface.</p>
        <div>
            <article>
                <img src="img/projectJohnDoe.PNG" alt="project visual">
                <div>
                    <h3>John Doe</h3>
                    <p>Intégration complète d'un portfolio de plusieurs pages, avec interface intéractive et animations vivantes</p>
                    <div><span>HTML</span><span>CSS</span><span>Responsive</span><span>Keyframes</span></div>
                    <a href="/john_doe/index.html" class="link" target="_blank">Voir le projet <i class="fa-solid fa-arrow-right"></i></a>
                    <a href="https://github.com/elise-kinn/John-Doe" class="link" target="_blank">Voir le repo GitHub <i class="fa-solid fa-arrow-right"></i></a>
                </div>
            </article>

            <article>
                <img src="img/projectPortfolio.png" alt="project visual">
                <div>
                    <h3>Portfolio</h3>
                    <p>Conception et intégration d'un portfolio d'une page, avec un design frais, une interface intéractive et un slider d'image</p>
                    <div><span>HTML</span><span>CSS</span><span>UI/UX Design</span><span>javaScript</span></div>
                    <a href="/portfolio/index.html" class="link" target="_blank">Voir le projet <i class="fa-solid fa-arrow-right"></i></a>
                    <a href="https://github.com/elise-kinn/HTML-CSS-Portfolio" class="link" target="_blank">Voir le repo GitHub <i class="fa-solid fa-arrow-right"></i></a>
                </div>
            </article>
        </div>
        <a href="projects.html" class="button blue">Voir tous mes projets</a>
    </div>
</section>
```

```css
@keyframes opacity {
    0% {opacity : 0}
    100% {opacity : 1}
}

#projects{
    padding : 65px 15px;
    background-color: var(--pale-rose);

    > div > p{
        text-align: center;
        margin-bottom : 50px;
    }

    > div > div{
        display: flex ;
        gap : 30px;
        max-width: 1300px;
        margin : auto;

        article{
            flex : 1;
            border-radius: 8px;
            box-shadow: 0px 4px 6px -1px #0000001A;
            padding-bottom : 15px;
            border : 2px dashed #9883928f;
```

```css
            &:hover{
                transform: rotate(-0.5deg);
                background-color: #e9dae9a0;

                img{
                    transform: scale(0.98);
                    filter: grayscale(20%)
                }

                h3::before, h3::after{
                    animation: opacity 0.2s ease forwards;
                }
            }

            > div{
                margin-left : 15px;

                h3{
                    font-size: 22px;
                    margin-top : 11px;
                    text-align: center;

                    &::after, &::before{
                        color : #83a206a1;
                        font-family:Impact, Haettenschweiler, 'Arial Narrow Bold', sans-serif;
                        font-weight: 700;
                        opacity : 0
                    }

                    &::before{
                        content : "</";
                        margin-right : 7px
                    }

                    &::after{
                        content : ">";
                        margin-left : 7px
                    }
                }
```

```css
                div{
                    margin : 14px 0 22px 0;
                    display: flex;
                    flex-wrap: wrap;
                    gap : 15px;

                    span{
                        background-color: var(--pink-span);
                        color : #77416d;
                        border-radius: 30px;
                        padding : 5px 10px;
                        font-size: 14px;

                        &:hover{
                            background-color: #6e416185;
                            color : #ffeefa;
                        }
                    }
                }

                img{
                    width : 100%;
                    border-radius:8px 8px 0 0;
                    height : 250px;
                }
            }

            >div > a{
                display: block;
                width: fit-content;
                margin: auto;
                margin-top : 40px
            }
        }
```

8. Section Contact

La section **#contact** contient un **formulaire statique** composé de :

- Deux <input type="text"> pour renseigner le nom et l'email
- Un <textarea> pour rédiger un message plus long
- Un <input type="checkbox" required"> pour valider le consentement RGPD
- Un **bouton** pour soumettre le formulaire

Mise en page

Pour organiser visuellement le formulaire, j'utilise display: grid avec grid-template-columns: 1fr 1fr.

La propriété grid-column permet de modifier la disposition des éléments au sein de la grille afin de mieux gérer leur placement.

Interactions Utilisateur

Pour supprimer l'effet de surbrillance des bordures lorsque l'on clique dans une zone de texte, j'ajoute : input: focus { outline : none }.

```
<section id="contact">
    <h2>Contactez-moi</h2>
    <p>Vous avez un projet en tête ou une question ? N'hésitez pas à me contacter, je <br>vous répondrai dans les plus brefs délais.</p>
    <form action="#">
        <label for="input-name">Nom</label>
        <label for="input-mail">Email</label>

        <input type="text" id="input-name" placeholder="Nom">
        <input type="text" id="input-mail" placeholder="Email">

        <label for="textarea-message">Message</label>
        <textarea name="" id="textarea-message" placeholder="Tapez votre message ici..."></textarea>


        <label for="input-checkbox"><input type="checkbox" id="input-checkbox" required> J'accepte que mes données soient traitées confor

        <button class="button green">Envoyer le message</button>
    </form>
</section>
```

```
#contact{
    padding : 50px 15px 70px 15px ;

    >p{
        text-align: center;
        margin-bottom : 50px;
    }

    form{
        display: grid;
        grid-template-columns: 1fr 1fr;
        max-width : 530px;
        margin : auto;
        gap : 5px;
        background-color: var(--pale-rose);
        padding : 40px;
        border-radius: 8px;
        border : 5px dashed #9883928f;

        label[for="textarea-message"],
        textarea[id="textarea-message"],
        label[for="input-checkbox"]{
            grid-column: 1/-1;
        }

        input[type="text"], textarea{
            padding : 10px 0 10px 10px;
            border-radius: 4px;
            border : 1px solid #442c3a57;
            margin-bottom : 13px;

            &:focus{
                outline: none;
            }
        }

        textarea{
            padding : 10px 0 70px 10px;
        }

        label[for="input-checkbox"]{
            margin-bottom : 10px;
        }

        button{
            border: none;
            width: fit-content;
            font-size: 15px;
        }
    }
}
```

9. Pages annexes

Pour que mes pages annexes aient une **hauteur supérieure à celle de n'importe quel écran**, j'ai utilisé la même technique que pour la section **#profile**, en ajoutant un **bandeau intégrant le titre de la page**.

```scss
#bandeau{
    position : relative;
    height : 360px;
    display: flex;
    align-items: center;
    justify-content: center;

    &:before{
        content :"";
        background-image: radial-gradient(#5d0046 2px, transparent 2px), radial-gradient(#5d0046 2px, transparent 2px);
        background-size: 91px 91px;
        background-position: 0 0, 45.5px 45.5px;
        background-color: #8a7482;
        /* padding : 150px 0; */
        width: 100%;
        height: 100%;
        position: absolute;
        z-index: -1;
    }

    h2{
        color : var(--pale-rose);
        font-size: 70px;
    }
}
```

10. Responsive

Le site ne nécessite pas de modification importante pour le **format tablette**. J'ai simplement :

- Ajusté la **photo de profil** et son ::after
- Réduit la taille de la **capture d'écran des projets** dans les articles de **#projects** pour un rendu plus harmonieux.

```scss
@media screen and (max-width: 1024px){ /*TABLETTE*/
    #profile > div{
        padding: 140px 15px;

        img {
            width : 200px
        }

        >div:after{
            right: 90px;
        }
    }

    #projects > div > div > article > img{
        height : 180px
    }
}
```

Pour le **format téléphone**, plusieurs adaptations sont nécessaires :

- Modification de la **barre de navigation** : disparition de la <nav> et affichage de l'**icône de menu**
- Réorganisation verticale des éléments utilisant display: flex, avec flex-direction: column
- Réduction du nombre de colonnes de la **grille des compétences**, passant de 3 à 2 colonnes sur 3 lignes pour un affichage plus lisible

```css
@media screen and (max-width: 670px){ /*MOBILE*/
    #nav{
        display: none;
    }

    #bars{
        display : block
    }

    #profile > div{
        flex-direction: column;

        img{
            margin-left :calc(50% - 100px) ;
        }

        h1 {
            font-size: 54px;
        }

        >div:after{
            right: -90px;
        }
    }

    #about{
        >div{
            flex-direction: column;

            > div > div{
                grid-template-columns: 1fr 1fr;
            }
        }
    }
```

```css
    #projects{
        > div > div{
            flex-direction: column;
        }
    }

    #contact form{
        grid-template-columns: 1fr;

        label[for="input-mail"]{
            grid-row:3 ;
        }
    }

    footer >div{
        flex-direction: column;

        #icone-container{

            width : fit-content;
            margin : 15px auto
        }

        p{
            text-align: left;
        }
    }
}
```

2. Précisez les moyens utilisés :

Google, Ordinateur, VS Code

3. Avec qui avez-vous travaillé ?

Seule

4. Contexte

Chantier, atelier, service	▸	Atelier
Période d'exercice	▸	**Du** 22/05/25 **au** 23/05/25

Exemple n°3 ▸ ***Maquetter des interfaces utilisateur web ou web mobile***

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

Description du projet : En s'appuyant sur des user stories, il faut créer une maquette qui illustre l'interface et le parcours utilisateur de l'application, en veillant à ce que chaque user story soit traitée et représentée visuellement, afin de couvrir l'ensemble des fonctionnalités attendues

1. User Stories

US-1 : Créer une fiche plante

En tant que collectionneur, je veux pouvoir ajouter une nouvelle plante à ma collection avec son nom, espèce, date d'acquisition et photo, afin de la cataloguer facilement.

US-2 : Consulter le détail d'une plante

En tant que collectionneur, je veux pouvoir consulter la fiche détaillée d'une plante (historique d'arrosage, de fertilisation, notes de croissance), afin de suivre son évolution dans le temps.

US-3 : Modifier ou supprimer une plante de ma collection

En tant que collectionneur, je veux pouvoir mettre à jour ou supprimer les informations d'une plante, afin de corriger ou retirer des plantes de ma collection.

US-4 : Recevoir des notifications d'entretien

En tant que collectionneur, je veux recevoir des notifications personnalisées pour l'arrosage et la fertilisation, afin de ne pas oublier les soins de mes plantes.

US-5 : Configurer les rappels d'entretien

En tant que collectionneur, je veux pouvoir personnaliser la fréquence et l'heure des rappels d'entretien pour chaque plante, afin d'adapter les notifications à leurs besoins spécifiques.

US-6 : Ajouter des photos d'évolution

En tant que collectionneur, je veux pouvoir ajouter régulièrement des photos à la fiche d'une plante, afin de documenter visuellement sa croissance au fil du temps.

US-7 : Filtrer ma collection de plantes

En tant que collectionneur, je veux pouvoir filtrer les plantes par type, besoin en lumière ou date d'acquisition, afin de mieux organiser ma collection.

US-8 : Trier ma collection de plantes

En tant que collectionneur, je veux pouvoir trier mes plantes par ordre alphabétique, date d'acquisition ou dernier entretien, afin de naviguer facilement dans ma collection.

US-9 : Visualiser un historique global d'entretien

En tant que collectionneur, je veux accéder à une page qui liste l'historique global des entretiens (arrosage, fertilisation, rempotage) pour toutes mes plantes, afin de suivre l'évolution de mes actions sur l'ensemble de ma collection.

2. Conception générale

Principes de design

La palette de couleur verte a été choisie car elle évoque la nature et l'univers des plantes, en cohérence avec le thème de l'application. L'architecture de navigation a été pensée pour être claire et ergonomique : chaque action est accessible en trois clics maximum. Les boutons de navigation assurent une utilisation simple et intuitive. Les textes sont conçus pour rester lisibles, avec un contraste suffisant et une taille de police minimale de 16 px..

Organisation générale

L'organisation générale de l'application est basique.

L'application repose sur une structure cohérente, avec un **header** et un **footer** présents sur toutes les pages.

- Le header intègre le logo (cliquable) et la barre de navigation principale.
- Le footer contient une navigation secondaire vers des pages complémentaires (non incluses dans cette maquette).

Les principales pages de l'application sont :

- **Page de collection** : écran d'accueil affichant toutes les fiches plantes créées par l'utilisateur. Un bandeau en haut permet de filtrer ou trier la collection. En bas de page, un bouton "Ajouter une plante" permet de créer une nouvelle fiche. Chaque fiche dispose d'actions "Modifier" et "Supprimer" (avec une modale de confirmation pour la suppression).
- **Page de création de fiche** : formulaire composé de quatre champs (nom, espèce, date d'acquisition, photo) et d'un bouton d'enregistrement.
- **Page de détails d'une plante** : affiche les informations détaillées et l'historique de la plante. Un formulaire intégré permet d'ajouter de nouvelles entrées à l'historique.
- **Page d'édition** : similaire à la création, mais le formulaire est pré-rempli avec les données existantes pour permettre la modification.
- **Page de paramétrage des notifications** : propose plusieurs interrupteurs (toggles) pour activer/désactiver les rappels, ainsi que des menus déroulants pour configurer fréquence et heure. Un bouton "Ajouter" permet de créer des notifications personnalisées.
- **Page historique global** : donne accès à l'historique complet des entretiens (arrosage, fertilisation, rempotage) de toutes les plantes. Un filtre permet de restreindre la vue à une plante spécifique.

3. Page "My Collection"

Objectif de la page

La page My Collection constitue l'écran principal de l'application. Elle permet d'avoir une vue d'ensemble rapide, tout en donnant un accès direct aux actions principales (consulter, modifier, supprimer, ajouter une plante).

Système de filtres et de tri

Un bandeau de contrôle est placé sous le titre afin de répondre aux user stories 7 et 8.

- **Filtres disponibles** : Besoins en lumière, Type de plante, Date d'acquisition
- **Tri** : Par date d'acquisition (par défaut)

Les filtres sont représentés sous forme de menus déroulants pour limiter l'encombrement visuel tout en restant facilement accessibles.

Affichage des fiches plantes

Les plantes sont présentées sous forme de **cartes**, un format choisi pour mettre en valeur la photo de chaque plante, et améliorer la lisibilité sur écran.

Chaque carte contient :

- Une **photo de la plante** (élément visuel central)
- Le **nom commun** et le **nom scientifique**
- La **date d'acquisition**
- La **date du dernier arrosage**, accompagnée d'une icône, pour une lecture rapide
- Un lien "*See more about my plant"* menant à la page de détails de la plante

Actions disponibles sur chaque carte (User Story 3)

Afin de répondre aux besoins de gestion, chaque fiche propose des actions directes :

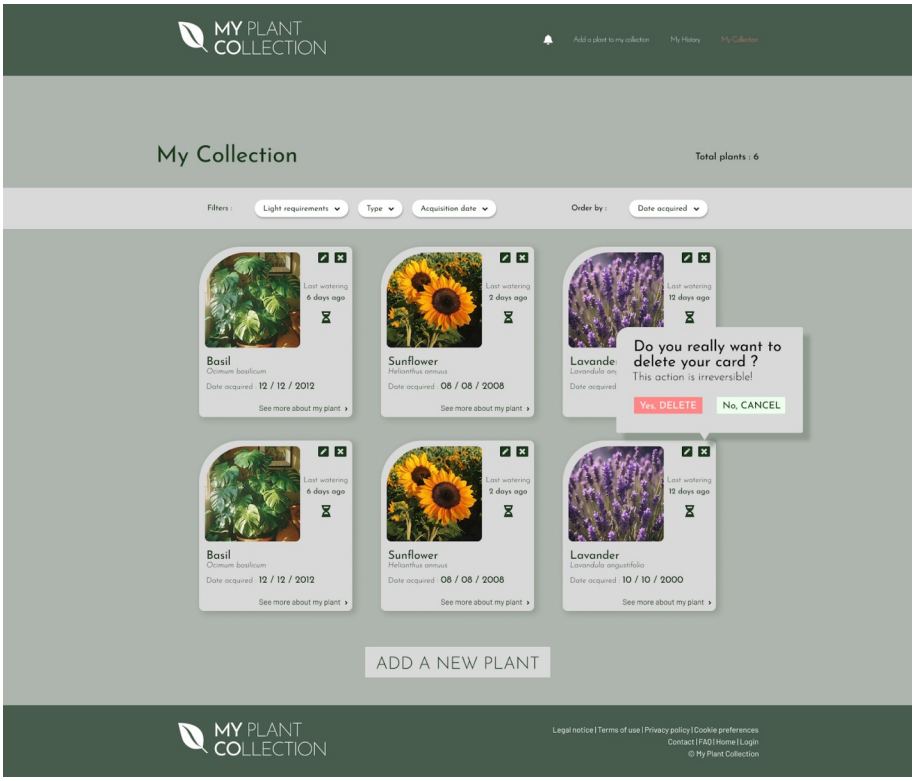
- **Modifier** : accès à la page d'édition avec formulaire pré-rempli
- **Supprimer** : ouverture d'une **modale de confirmation**

La modale de suppression a été intégrée pour éviter les suppressions accidentelles, et ainsi informer clairement l'utilisateur du caractère irréversible de l'action.

Ajout d'une nouvelle plante (User Story 1)

En bas de la page, un bouton bien visible “**Add a new plant”** permet d’accéder à la page de création.

Le bouton se distingue visuellement tout en restant cohérent avec la palette de couleurs de l’application.



4. Page “Plant Card”

Objectif de la page

La page **Détail de la plante** permet à l’utilisateur de consulter l’ensemble des informations relatives à une plante spécifique et de suivre son entretien dans le temps. Elle répond notamment aux user stories 2 et 6.

Navigation et contexte

Un bouton de retour placé à côté du titre permet de revenir rapidement à la page de collection.
Le titre “Plant card : *Nom de la plante*” rappelle clairement la plante actuellement consultée, évitant toute confusion lorsque l’utilisateur navigue entre plusieurs fiches.

Informations principales

La partie gauche de l’écran est dédiée aux **informations générales de la plante**, regroupées dans une carte

visuelle cohérente avec celles utilisées sur la page collection.

Cette carte contient :

- Une **photo principale** de la plante, occupant une place centrale pour renforcer l'identification visuelle
- Le **nom commun** et le **nom scientifique**
- La **date d'acquisition**
- Des informations complémentaires : Indice de croissance, Type de fertilisation, Date du dernier arrosage

Ce regroupement permet une lecture rapide des données essentielles sans surcharge cognitive.

Également, les icônes d'actions (Modifier et Supprimer) sont positionnées en haut de la page.

Ajout d'une nouvelle entrée à l'historique (User Story 6)

La partie droite de la page propose un bloc "**New entry"** permettant d'enrichir l'historique de la plante. Ce formulaire comprend :

- Un menu déroulant pour sélectionner le **type d'entrée** (arrosage, fertilisation, rempotage, etc.)
- Un champ pour **ajouter une photo**
- Un bouton "**Add"** pour valider l'ajout

Ce choix d'un formulaire intégré évite de changer de page et favorise une interaction rapide et fluide.

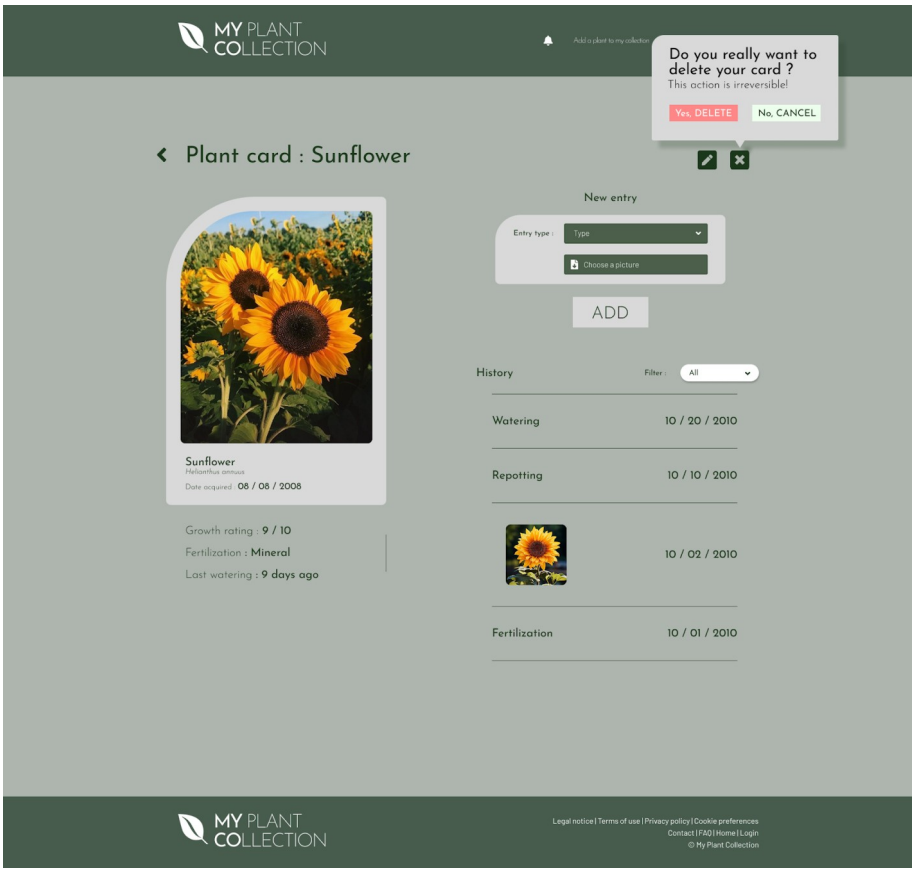
Historique de la plante (User Story 9)

Sous le formulaire, un espace est consacré à l'**historique des actions** effectuées sur la plante. Les entrées sont affichées de manière chronologique Chaque action indique :

- Le type d'entretien
- La date correspondante
- Une image associée, lorsque disponible
-

Un **filtre** permet d'afficher toutes les actions ou de restreindre la vue à un type spécifique.

Cette section répond au besoin de suivi à long terme et permet à l'utilisateur d'analyser l'entretien effectué sur la plante.



5. Page “Add Plant”

Objectif de la page

La page **Ajouter une plante** permet à l’utilisateur de créer une nouvelle fiche plante et de l’intégrer à sa collection. Elle répond à la user story 1. Cette page est conçue pour rendre la saisie simple, rapide et sans surcharge d’informations.

Navigation et contexte

- Un **bouton de retour** est placé en haut de la page afin de permettre à l’utilisateur de revenir facilement à la page précédente sans perdre le contexte.
- Le titre indique clairement l’action en cours.
- Un texte descriptif accompagne le titre pour guider l’utilisateur et expliquer l’objectif du formulaire.

Structure du formulaire

Le formulaire est centré sur la page et présenté dans un **conteneur visuel distinct**, afin de focaliser

l'attention de l'utilisateur sur l'action principale. Il se compose de **quatre champs** :

- **Nom commun** (obligatoire)
- **Nom scientifique** (facultatif)
- **Date d'acquisition**
- **Photo de la plante**

Les champs obligatoires sont clairement indiqués par un astérisque, ce qui permet d'éviter les erreurs de saisie.

Choix des composants UI

- Les champs texte utilisent des **inputs simples et lisibles**, avec des libellés explicites.
- La date d'acquisition est sélectionnée via un **sélecteur de date**, limitant les erreurs de format.
- Le champ photo repose sur un bouton "*Choose a picture"*, offrant une interaction claire et familière pour l'utilisateur.
- L'ensemble respecte une taille de police minimale assurant une bonne lisibilité.
- Un bouton principal "**Add my plant"** est placé sous le formulaire. Il est visuellement distinct pour signaler l'action finale. Son libellé explicite confirme l'intention de l'utilisateur avant la création de la fiche.

6. Page “Edit a plant”

Objectif de la page

La page Éditer une plante permet à l’utilisateur de modifier les informations d’une fiche plante existante. Elle répond à la user story 3. Cette page reprend volontairement la structure de la page de création afin de garantir une continuité d’usage.

Formulaire pré-rempli

Le cœur de la page repose sur un formulaire identique à celui de la création, mais avec des champs pré-remplis à partir des données existantes de la plante.

Les champs disponibles sont :

- Nom commun (obligatoire)
- Nom scientifique
- Date d’acquisition
- Photo

Le pré-remplissage permet de réduire le temps de saisie, d’éviter la ressaisie inutile, ainsi que de limiter les erreurs de modification.

Gestion de la photo

La photo actuelle de la plante est affichée sous le champ image afin de donner un repère visuel immédiat. Un bouton “Choose to change picture” permet de remplacer l’image existante. Une icône de suppression permet, le cas échéant, de retirer la photo actuelle.

Ce fonctionnement offre de la flexibilité tout en restant simple à comprendre.

Choix UX et cohérence globale

- Le bouton principal “Edit my plant” valide les modifications effectuées.
- La réutilisation des mêmes composants que la page de création permet une prise en main immédiate.
- Les différences visuelles se limitent au strict nécessaire (titre, pré-remplissage, libellés), afin de ne pas perturber l’utilisateur.
- Cette page s’inscrit naturellement dans le parcours : Collection → Détail → Édition → Retour au détail ou à la collection.

7. Page “My History”

Objectif de la page

La page Historique global permet à l’utilisateur de consulter l’ensemble des actions réalisées sur toutes les plantes de sa collection, en un seul endroit. Elle répond à la user story 9. Cette page est pensée comme un outil de suivi et de synthèse.

Structure de l’historique

L’historique est présenté sous la forme d’une liste structurée, facilitant la lecture et la comparaison des entrées. Chaque ligne comprend :

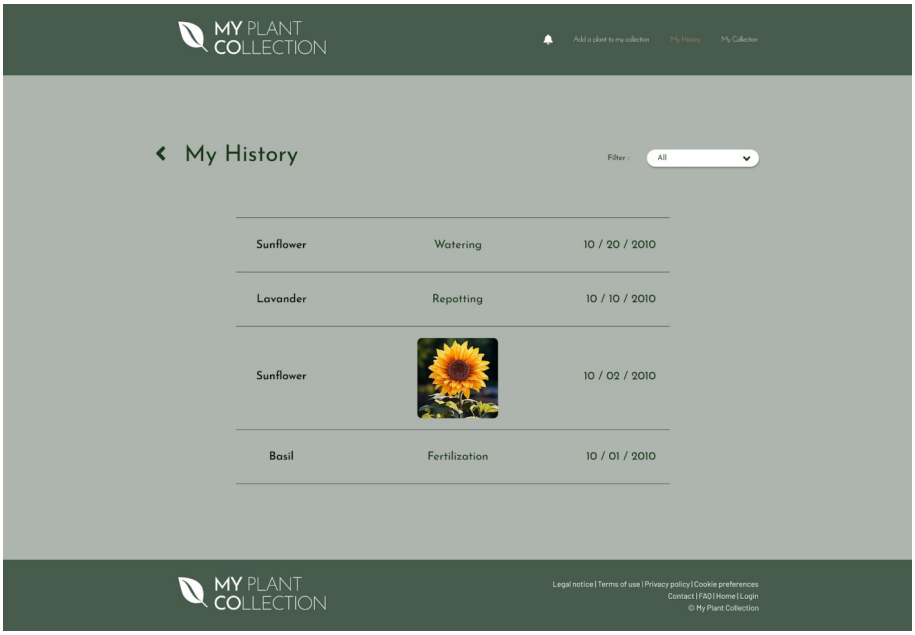
- Le nom de la plante
- Le type d’action effectuée (arrosage, rempotage, fertilisation, etc.)
- La date de l’action
- Une image associée, lorsqu’elle est disponible

Cette présentation permet de repérer rapidement les interventions réalisées, tout en conservant une hiérarchie visuelle simple.

Système de filtrage

Un menu déroulant de filtre est placé en haut de la page. Il permet d’afficher toutes les entrées ou de restreindre l’historique selon un critère spécifique. Ce filtre améliore la lisibilité lorsque le volume d’entrées

devient important.



8. Page “Notification settings”

Objectif de la page

La page Notification settings permet à l’utilisateur de gérer les rappels liés à l’entretien de ses plantes. Elle répond à la user story 4 et 5.

Activation globale des notifications

Un interrupteur principal “Enable notifications” permet d’activer ou de désactiver l’ensemble des rappels. Ce choix offre un contrôle global et évite à l’utilisateur de devoir gérer chaque rappel individuellement s’il ne souhaite pas recevoir de notifications.

Types de rappels disponibles

La page propose plusieurs catégories de rappels liées à l’entretien des plantes :

- Arrosage
- Fertilisation
- Rempotage

Chaque catégorie dispose de :

- un interrupteur ON/OFF permettant d’activer ou non le type de rappel

- une configuration indépendante, garantissant une grande flexibilité

Paramétrage par plante

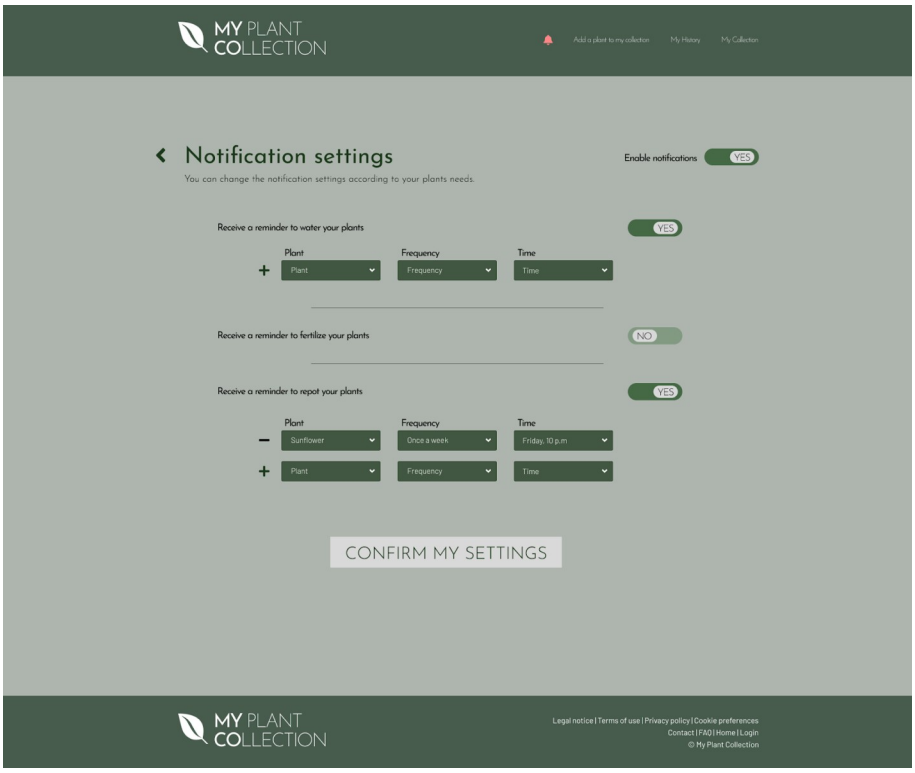
Pour les rappels activés, l'utilisateur peut définir :

- La plante concernée
- La fréquence du rappel
- L'heure de notification

Un bouton "+" permet d'ajouter plusieurs rappels pour différentes plantes au sein d'un même type d'entretien, et un bouton "-" permet de supprimer une ligne existante, offrant un contrôle simple et intuitif.

Choix UX et cohérence

- Les interrupteurs offrent un retour visuel immédiat sur l'état des notifications.
- La mise en page aérée facilite la lecture et la compréhension des options.
- Les composants (boutons, listes déroulantes, couleurs) sont cohérents avec le reste de l'application, assurant une continuité graphique.



9. Conclusion

Ce projet propose une application claire et cohérente dédiée à la gestion et au suivi des plantes.
Grâce à une interface intuitive et homogène, l'utilisateur peut facilement ajouter, consulter, modifier et entretenir sa collection au quotidien.
Les différentes fonctionnalités, comme l'historique global et les notifications personnalisées, renforcent l'accompagnement sur le long terme.
L'ensemble du parcours utilisateur a été pensé pour être simple, fluide et accessible, tout en répondant aux besoins essentiels de suivi et d'organisation.

2. Précisez les moyens utilisés :

Google, Figma

3. Avec qui avez-vous travaillé ?

Seule

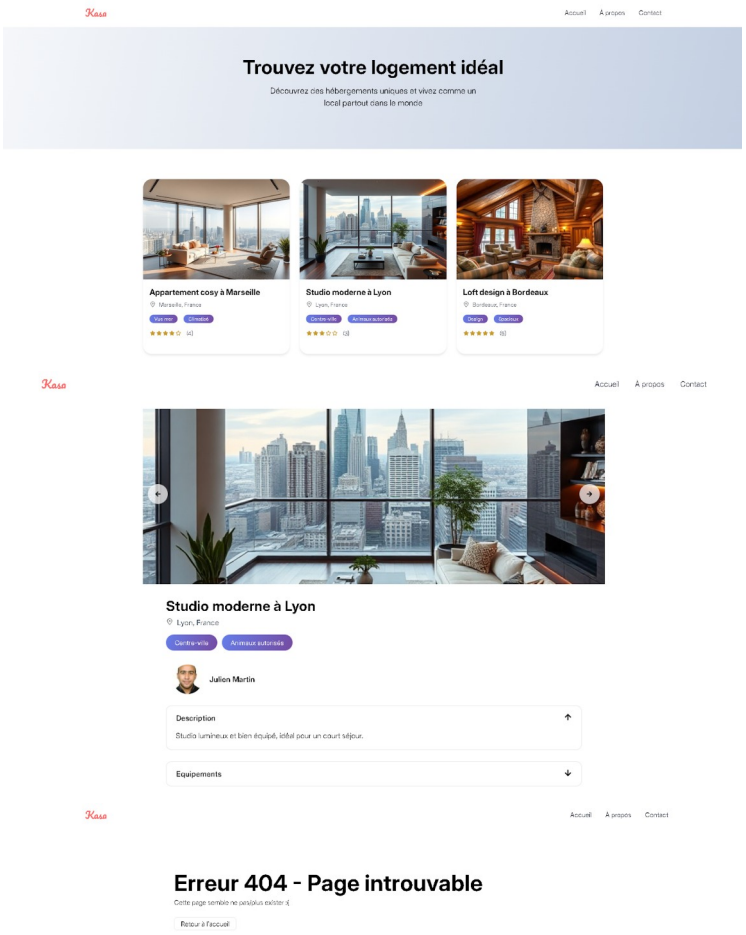
4. Contexte

Nom de l'entreprise, organisme ou association	**Centre de formation La Plateforme**
Chantier, atelier, service	Atelier
Période d'exercice	**Du** 23/07/25 **au** 25/07/25

Exemple n°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 :

Description du projet : Création d'une application React avec Vite, TypeScript, Sass qui simule un site de location de logements ou d'hébergements, type "Airbnb simplifié".



1. Création du projet

Pour initier ce projet React, j'ai commencé par créer un dossier `projet-react` qui sert d'espace de travail. Dans le terminal, j'ai exécuté la commande suivante pour installer la base du projet : `npm create vite@latest`. J'ai ensuite indiqué le nom du projet, puis sélectionné les options : React comme framework et TypeScript comme langage.

Une fois le projet généré, j'ai installé les dépendances nécessaires :

- `npm install react-router-dom` : Gestion des routes
- `npm install sass` : meilleure organisation et modularisation du CSS
- `npm install react-icons` : utilisation d'icônes via la librairie React Icons.

À ce stade, l'architecture du projet ressemble à ceci :

Enfin, j'ai créé un repository GitHub et relié le projet afin d'assurer le versioning et la sauvegarde du code.

2. Initialisation

Avant de commencer le développement, j'ai organisé la structure du projet. Dans le dossier src, j'ai créé plusieurs sous-dossiers pour mieux séparer les différentes parties de l'application :

- `components` pour les composants React
- `data` pour le fichier JSON
- `pages` pour les pages React
- `styles` pour les fichiers SCSS

J'ai également supprimé certains fichiers générés automatiquement qui n'étaient pas utiles, tels que App.css, index.css, vite.svg et react.svg.

Puis, j'ai modifié main.tsx et App.tsx pour ajouter la gestion des Routes et les routes pour chaque

page (notFound, home, logement). Comme le header et le footer seront présents dans toute l'application, j'insère leurs composants ici.

```jsx
import React from 'react'
import ReactDOM from 'react-dom/client'
import App from './App.tsx'
import { BrowserRouter } from 'react-router-dom'
import './styles/global.scss'

ReactDOM.createRoot(document.getElementById('root')!).render(
  <React.StrictMode>
    <BrowserRouter>
      <App />
    </BrowserRouter>
  </React.StrictMode>,
)
```

```jsx
function App() {
  return (
    <>
    <Header />

    <Routes>
      <Route path='/' element={<Home/>} />
      <Route path='/logement/:id' element={<Logement />}
      <Route path='*' element={<NotFound />} />
    </Routes>

    <Footer />
    </>
  )
}
```

Enfin, dans le fichier index.html, j'ai modifié le code afin de supprimer l'icône par défaut et de mettre à jour le titre de l'application.

3. home.tsx

```jsx
import Hero from "../components/unique/hero"
import CardSection from "../components/unique/cardSection"

const Home = () => {
    return (
        <>
            <Hero />
            <CardSection />
        </>
    )
}

export default Home
```

J'ai commencé le développement par la page Home (home.tsx). Selon la maquette Figma fournie, elle se compose d'une section Hero et d'un affichage de logements sous forme de cartes (cards). Pour garder un code clair et modulable, j'ai créé deux composants distincts : Hero et CardSection.

hero.tsx : renvoie une section simple avec une div contenant deux textes.

```jsx
const Hero = () => {
    return(
        <section id="hero">
            <div>
                <h1>Trouvez votre logement idéal</h1>
                <p>Découvrez des hébergements uniques et vivez comme un local partout dans le monde</p>
            </div>
        </section>
    )
}

export default Hero
```

cardSection.tsx : renvoie une section contenant une boucle map(). Celle-ci parcourt le JSON (importé à la ligne 2) qui regroupe les informations des logements, et affiche chaque élément à l'aide du composant Card, en lui transmettant : img, title, location, url, tags et rating.

```tsx
import Card from "../repeat/card"
import data from "../../data/logements.json";

const CardSection = () => {
    return(
        <section id="card-section">
            {data.map(logement => (
                    <Card
                        img={logement.pictures[0]}
                        title={logement.title}
                        location={logement.location}
                        url={`/logement/${logement.id}`}
                        tags={logement.tags}
                        rating={logement.rating}
                    />
                ))}
        </section>
    )
}

export default CardSection
```

card.tsx : utilise les données reçues pour générer un lien (<Link> de react-router-dom). La propriété to={url} permet de construire une URL de type /logement/${logement.id}. Ainsi, au clic, l'utilisateur est redirigé vers la page détaillée du logement correspondant.

Comme on est en TypeScript, les données reçues sont typées via CardProp, ce qui permet de définir précisément la nature de chaque variable.

Ce composant contient également deux sous-composants :

- Stars : gère l'affichage dynamique des étoiles selon la note,
- ArticleInfo : affiche des informations complémentaires, réutilisables dans d'autres pages.

```tsx
import { Link } from 'react-router-dom'
import Stars from './stars';
import ArticleInfo from './articleInfo';

type CardProp = {
    img: string,
    title: string,
    location: string,
    url: string,
    tags: string[],
    rating: string
}
```

```tsx
const Card = ({img, title, location, url, tags, rating}: CardProp) => {
    return (
        <Link to={url}>
            <article className='card-article'>
                <img src={img} />
                <div>
                    <ArticleInfo title={title} location={location} tags={tags}/>
                    <div className='rating-div'>
                        <Stars rating={parseInt(rating)}/>
                        <p>({rating})</p>
                    </div>
                </div>
            </article>
        </Link>
    )
}

export default Card
```

articleInfo.tsx : renvoie une simple div contenant les données transmises par son parent. Une icône est importée à la ligne 1 et utilisée à la ligne 15. Les données sont typées via ArticleInfoProp.

Ce composant inclut un sous-composant Tags, qui génère un span pour chaque élément du tableau tags (type string[]), grâce à une boucle map().

```tsx
import { CiLocationOn } from "react-icons/ci";
import Tag from './tag';

type ArticleInfoProp = {
    title: string,
    location: string,
    tags: string[]
}

const ArticleInfo = ({title, location, tags}: ArticleInfoProp) => {
    return (
        <div className="article-info-div">
            <h2>{title}</h2>
            <div className="loc-div">
                <CiLocationOn />
                <p>{location}</p>
            </div>
            <div className='tags-div'>
                {tags.map((el, i) =>
                    <Tag text={el} key={i}/>
                )}
            </div>
        </div>
    )
}

export default ArticleInfo
```

```tsx
type TagProp = {
    text: string
}

const Tag = ({ text }: TagProp) => {
    return(
        <span className='span-tag'>{text}</span>
    )
}

export default Tag
```

stars.tsx : crée un tableau de longueur 5 et boucle dessus. Pour chaque élément, l'index est comparé à la note (rating) du logement :

- Si l'index est supérieur à la note → étoile vide
- Sinon → étoile remplie

```tsx
import { TiStarFullOutline } from "react-icons/ti";
import { TiStarOutline } from "react-icons/ti";

type starsProp = {
    rating: number,
}

const Stars = ({ rating }: starsProp) => {
    return(
        <>
        {Array.from({ length: 5 }, (_, i) =>
        i < rating ? <TiStarFullOutline key={i} /> : <TiStarOutline key={i} />
      )}
        </>
    )
}

export default Stars
```

4. logement.tsx

```tsx
import { useParams } from "react-router-dom";

import ArticleInfo from "../components/repeat/articleInfo";
import Collapse from "../components/repeat/collapse";
import Equipment from "../components/repeat/equipment";
import Carousel from '../components/unique/carousel';
import ErrorComp from '../components/repeat/errorComp';

import data from "../data/logements.json"

type LogementProp = {
  id: string,
  title: string,
  location: string,
  pictures: string[],
  host: { name: string; picture: string },
  description: string[],
  equipments: string[],
  tags: string[]
}

const logements: LogementProp[] = data as unknown as LogementProp[];

const Logement = () => {
    const { id } = useParams<{ id: string; }>();

    const logementData = logements.find(el => el.id === id);

    if (!logementData) return ( // si id introuvable
        <ErrorComp title="Logement introuvable :(" subtitle="Le logement que vous cherchez semble ne pas/plus exister."/>

    );
```

```tsx
    return ( //sinon
        <>
            <Carousel imgs={logementData.pictures}/>

            <section id="info">
                <ArticleInfo
                    title={logementData.title}
                    location={logementData.location}
                    tags={logementData.tags}
                />
                <div id="host">
                    <img src={logementData.host.picture} alt="Host pic" />
                    <h2>{logementData.host.name}</h2>
                </div>
            </section>

            <Collapse title="Description" children={logementData.description}/>
            <Collapse title="Equipements" children={<ul id="equipment">{logementData.equipments.map((el) => <Equipment text={el}/>)}</ul>}/>

        </>
    );
};
```

La page Logement affiche les détails d'un logement en fonction de son id. Dans App.tsx, j'ai créé une route spécifique : path = '/logement/:id'.

L'id transmis dans l'URL par le <Link> est récupéré grâce au hook useParams() de react-router-dom.

```tsx
const { id } = useParams<{ id: string; }>();
```

À partir de cet id, j'utilise la méthode find() pour rechercher dans le JSON le logement correspondant.

```tsx
const logementData = logements.find(el => el.id === id);
```

```tsx
import HomeBtn from "./homeBtn";

type ErrorCompProp = {
    title: string,
    subtitle: string
}

const ErrorComp = ({ title, subtitle }: ErrorCompProp) => {
    return(
        <section id="not-found">
            <h1>{title}</h1>
            <p>{subtitle}</p>
            <HomeBtn />
        </section>
    )
}

export default ErrorComp
```

```tsx
import { Link } from 'react-router-dom'

const HomeBtn = () => {
    return <Link to="/">Retour à l'accueil</Link>
}

export default HomeBtn
```

- Si l'id ne correspond à aucun logement, le composant ErrorComp affiche une erreur 404 avec un bouton permettant de revenir à l'accueil
- Si l'id est valide, la page affiche :
    - un Carrousel
    - une section d'informations (incluant le composant ArticleInfo)
    - deux composants Collapse.

```tsx
import { useState } from 'react'

import { FaArrowLeft } from "react-icons/fa";
import { FaArrowRight } from "react-icons/fa";

type CarouselProp = {
    imgs: string[]
}

const Carousel = ({ imgs }: CarouselProp) => {
    const [ index, setIndex ] = useState(0)

    const next = () => setIndex((index + 1) % imgs.length)
    const prev = () => setIndex((index - 1 + imgs.length) % imgs.length)

    return(
        <section id="carousel">
            <button className='carousel-btn' id="carousel-prev" onClick={prev}><FaArrowLeft /></button>
            <img src={imgs[index]} alt="carousel" />
            <button className='carousel-btn' id="carousel-next" onClick={next}><FaArrowRight /></button>
        </section>
    )
}

export default Carousel
```

carrousel.tsx : permet d'afficher un diaporama des photos du logement, avec deux boutons *Suivant* et *Précédent*.

J'importe et initialise useState() pour gérer l'index courant :

```
const [ index, setIndex ] = useState(0)
```

Deux fonctions modifient l'index au clic sur les boutons :

- (index + 1) % longueurDuTableau → empêche de dépasser la fin du tableau,
- (index - 1 + longueurDuTableau) % longueurDuTableau → empêche de descendre en dessous de 0.

La section contient ensuite :

- deux boutons avec onClick={fonction}
- une image dont la src varie selon la valeur de index

```
import { useState } from 'react'; // pour déclarer une variable d'état

import { FaArrowDown } from "react-icons/fa";
import { FaArrowUp } from "react-icons/fa";

type CollapseProp = {
    title: string,
    children: React.ReactNode
}

const Collapse = ({ title, children }: CollapseProp) => {
    const [ open, setOpen ] = useState(false) // déclaration de la variable d'état useState(initialState)

    return(
        <section className='collapse'>
            <div>
                <button onClick={ () => setOpen(!open)}>
                    {title} { open ? <FaArrowUp /> : <FaArrowDown /> }
                </button>
                {open && <div className='collapse-content'>{children}</div>}
            </div>
        </section>
    )

}

export default Collapse
```

collapse.tsx : gère l'ouverture et la fermeture d'une section au clic.

J'importe et initialise un boolean avec useState() :

```
const [ open, setOpen ] = useState(false)
```

Au clic sur le bouton, l'état est inversé :

```
<button onClick={ () => setOpen(!open)}>
```

Une div s'affiche uniquement si open est true (grâce à &&).

```
{open && <div className='collapse-content'>{children}</div>}
```

Le deuxième Collapse contient en plus une boucle map() qui génère une liste d'équipements à partir du tableau fourni par le JSON.

```tsx
import { FaPlusSquare } from "react-icons/fa";

type EquipmentProp = {
    text: string,
}

const Equipment = ({ text }: EquipmentProp) => {
    return(
        <li> <FaPlusSquare /> {text}</li>
    )
}
export default Equipment
```

5. notFound.tsx

Lorsque l'URL ne correspond à aucune route définie dans App.tsx, c'est la page NotFound qui est affichée.

```tsx
<Route path='*' element={<NotFound />} />
```

```tsx
import ErrorComp from "../components/repeat/errorComp"

const NotFound = () => {
    return(
        <>
        <ErrorComp title="Erreur 404 - Page introuvable" subtitle="Cette page semble ne pas/plus exister :("/>
        </>
    )
}
export default NotFound
```

Cette page renvoie simplement le composant ErrorComp (présenté plus haut), qui gère l'affichage d'une erreur 404 et propose de retourner à la page d'accueil.

2. Précisez les moyens utilisés :

Ordinateur, VS Code, Google

3. Avec qui avez-vous travaillé ?

Seule

4. Contexte

Chantier, atelier, service	Atelier
Période d'exercice	**Du** 30/07/25 **au** 01/08/25

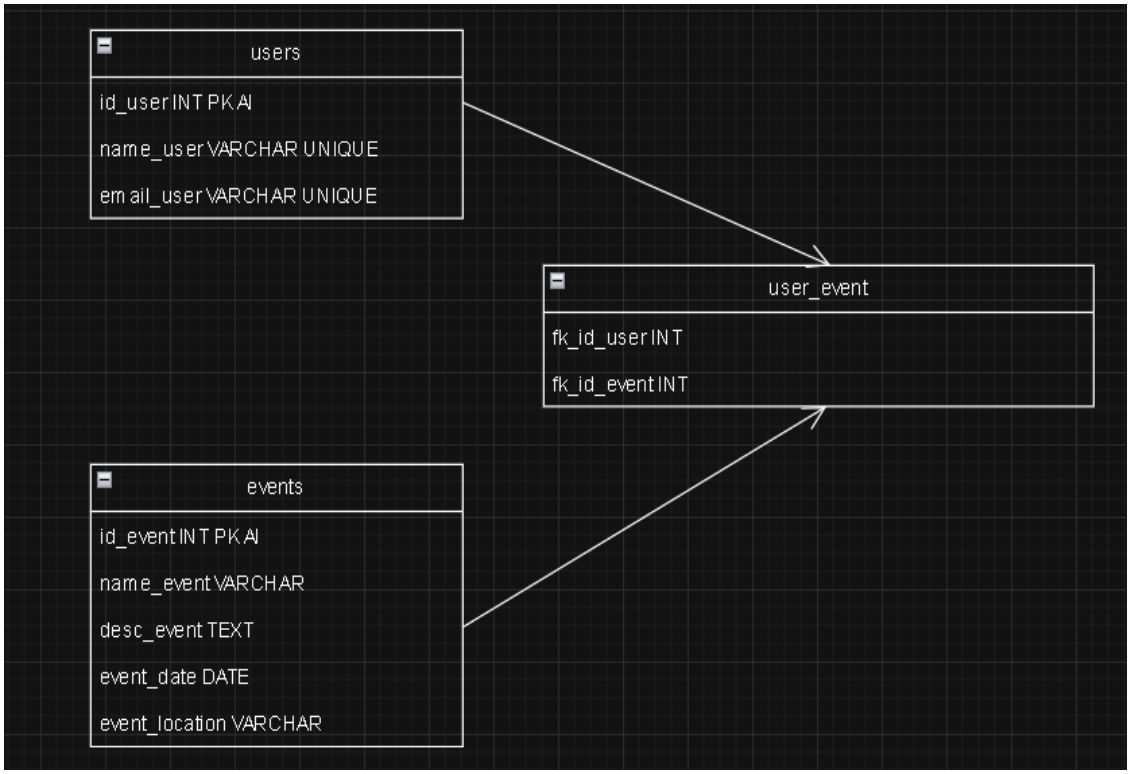
Activité-type 2 — **Développer la partie back-end d'une application web ou web mobile sécurisée**

Exemple n°1 ▸ ***Mettre en place une base de données relationnelle***

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

Description du projet : Conception et implémentation d'une base de donnée relationnelle en langage SQL pour une plateforme en ligne de gestion d'évènements, qui stockera de manière structurée et fiable toutes les informations essentielles (utilisateurs, inscriptions, évènements)

1. Diagramme relationnel



Le diagramme ci-dessus représente le schéma relationnel de la base de données.

- Table **Utilisateur** : chaque utilisateur dispose d'un identifiant unique, d'un nom d'utilisateur (unique) et d'un e-mail (également unique).
- Table **Événement** : un organisateur peut créer un événement en renseignant son nom, une description, la date et le lieu.
- Table de jonction **Inscription** : grâce à cette relation de type *n-n* (plusieurs utilisateurs peuvent participer à plusieurs événements), les inscriptions sont gérées de manière centralisée. Cette table contient les clés étrangères reliant Utilisateur et Événement.

2. Conception de la base de donnée

```sql
--Création de la base de donnée
CREATE DATABASE IF NOT EXISTS `events`;
USE `events`;

--Création de la table utilisateurs (nom / email)
CREATE TABLE users (
    id_user INT PRIMARY KEY AUTO_INCREMENT,
    username VARCHAR(255) NOT NULL,
    email VARCHAR(100) NOT NULL,
    UNIQUE (username, email)
)Engine=InnoDB;

--Création de la table évènements (nom / description / date / lieu)
CREATE TABLE events (
    id_event INT PRIMARY KEY AUTO_INCREMENT,
    event_name VARCHAR(100) NOT NULL,
    event_desc TEXT NOT NULL,
    event_date DATE NOT NULL,
    event_location VARCHAR(255) NOT NULL
)Engine=InnoDB;

--Création de la table de jointure utilisateur - évènement (foreign keys x 2)
CREATE TABLE user_event(
    fk_id_user INT,
    fk_id_event INT,
    PRIMARY KEY (fk_id_user, fk_id_event),
    FOREIGN KEY (fk_id_user)
        REFERENCES users(id_user)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY (fk_id_event)
        REFERENCES events(id_event)
        ON DELETE CASCADE
        ON UPDATE CASCADE
)Engine=InnoDB;
```

Table Users

La table **USERS** contient trois colonnes :

- **id_user** : clé primaire auto-incrémentée, de type INT, permettant d'identifier chaque utilisateur de manière unique.
- **username** et **email** : deux colonnes de type VARCHAR, définies avec les contraintes UNIQUE et NOT NULL afin d'empêcher la duplication et de garantir qu'aucun compte ne soit créé sans ces informations essentielles. La longueur maximale est précisée entre parenthèses.

Table Events

La table **EVENTS** contient cinq colonnes :

- **id_event** : clé primaire auto-incrémentée, de type INT, permettant d'identifier chaque événement.
- **event_name**: de type VARCHAR(255), non nul.
- **event_desc**: de type TEXT, pour accueillir un contenu potentiellement long.
- **event_date**: de type DATE, stockant l'année, le mois et le jour.

- **event_location**: de type VARCHAR(255).

Toutes les colonnes descriptives sont définies avec NOT NULL pour éviter la création d'événements incomplets.

Table Inscriptions (table de jointure)
Cette table assure la relation *n-n* entre **USERS** et **EVENTS**. Elle est composée de :

- **user_id** : clé étrangère référençant USERS(id).
- **event_id** : clé étrangère référençant EVENTS(id).

Les deux colonnes forment une clé primaire composite, ce qui empêche la duplication d'inscriptions (un même utilisateur ne peut s'inscrire deux fois au même événement).
Les contraintes ON DELETE CASCADE et ON UPDATE CASCADE garantissent que la suppression ou la modification d'un utilisateur ou d'un événement entraîne automatiquement la mise à jour de cette relation.

Moteur de stockage
La formule **ENGINE=InnoDB** est utilisée, car ce moteur gère les clés étrangères et permet de maintenir l'intégrité référentielle.

3. Requêtes SQL

US1 : Créer un compte utilisateur

```
INSERT INTO users(username, email) VALUES ('Elise', 'elise@ex.com');
```

Cette requête permet d'insérer un nouvel utilisateur dans la table **users**.
La syntaxe INSERT INTO table(col1, col2) VALUES ('valeur1', 'valeur2') ajoute une nouvelle ligne avec les valeurs indiquées.

US2 : Consulter les événements disponibles

```
SELECT * FROM events;
```

Cette requête affiche l'ensemble des événements présents dans la table **events**.
La commande SELECT permet de récupérer des données : l'astérisque * indique que toutes les colonnes doivent être affichées.

US3 : S'inscrire à un événement

INSERT INTO user_event(fk_id_user, fk_id_event) VALUES (5, 1);

Cette requête ajoute une nouvelle ligne dans la table de jointure **user_event**, en reliant l'utilisateur (id = 5) à l'événement (id = 1).

La syntaxe est similaire à celle de l'US-1, mais appliquée à la table de relation.

US4 : Consulter mes inscriptions personnelles

```
SELECT event_name, event_desc, event_date, event_location
FROM user_event
LEFT JOIN events ON fk_id_event = id_event
WHERE fk_id_user = 2;
```

Cette requête permet à un utilisateur de visualiser uniquement ses inscriptions.

- LEFT JOIN relie la table **user_event** à la table **events** en fonction de l'id de l'événement.
- La clause WHERE limite l'affichage aux inscriptions correspondant à un utilisateur précis (ici fk_id_user = 2).
- Seules certaines colonnes de **events** sont sélectionnées pour présenter les informations pertinentes de l'événement

US5 : Créer un nouvel événement

INSERT INTO events(event_name, event_desc, event_date, event_location) VALUES ('La Folie', 'Un évènement de folie', '2025-07-28', 'Stade de France, Paris');

Cette requête insère un nouvel événement dans la table **events**.

La syntaxe INSERT INTO est la même que pour les utilisateurs, mais appliquée aux colonnes d'événement.

US6 : Lister les participants d'un événement

```
SELECT username, email
FROM user_event
LEFT JOIN users ON fk_id_user = id_user
WHERE fk_id_event = 2;
```

Cette requête permet d'afficher la liste des participants d'un événement spécifique.

- La clause LEFT JOIN relie la table **user_event** à la table **users**.
- La condition WHERE filtre uniquement les inscriptions liées à l'événement id = 2.

US7 : Analyser la popularité des événements

```
SELECT event_name, COUNT(*) AS count_subscription
```

FROM user_event

LEFT JOIN users ON fk_id_user = id_user

GROUP BY id_event

Cette requête retourne le nombre de participants par événement.

- La fonction COUNT() compte le nombre d'inscriptions.
- La clause GROUP BY regroupe les résultats par identifiant d'évènement.

US8 : Lister tous les utilisateurs enregistrés

SELECT * FROM users

Cette requête affiche l'intégralité des utilisateurs inscrits dans la table **users**.

L'astérisque * sélectionne toutes les colonnes disponibles.

US9 : Mettre à jour les informations d'un événement

UPDATE events SET event_location = 'Stade De France' WHERE event_location = 'Sade De France'

Cette requête corrige l'adresse d'un événement.

- La commande UPDATE modifie une ou plusieurs colonnes.
- La clause WHERE permet de cibler uniquement les lignes concernées.

US10 : Annuler son inscription à un événement

DELETE FROM user_event WHERE fk_id_user = 2 AND id_event = 2

Cette requête supprime l'inscription d'un utilisateur à un événement donné.

- La commande DELETE supprime une ligne.
- La clause WHERE identifie précisément l'utilisateur et l'événement concerné.

US11 : Supprimer un événement annulé

DELETE FROM events WHERE id_event = 2

Cette requête supprime un événement de la table **events** en fonction de son identifiant.

Grâce à la contrainte ON DELETE CASCADE, les inscriptions liées à cet événement seront automatiquement supprimées de la table de jointure.

4. Mise en place de la base de données

Pour mettre en place la base de données relationnelle :

1. Lancement du serveur local :

- Démarrer le serveur local avec WAMP.
- Accéder à l'interface de gestion via l'URL : http://localhost/phpmyadmin
- Se connecter avec le compte administrateur par défaut (root), qui ne possède pas de mot de passe.

2. Création de la base et des tables (DDL = Data Definition Language) : Dans l'onglet **SQL**, saisir le code DDL (CREATE DATABASE, CREATE TABLE) afin de définir la base de données et créer les tables vides avec leurs contraintes.

3. Insertion des premières données (DML = Data Manipulation Language) : Toujours dans l'onglet **SQL**, entrer les instructions INSERT INTO pour ajouter des enregistrements et peupler la base (ex. : utilisateurs et événements).

4. Manipulation et gestion des données (Requêtes SQL) : Enfin, exécuter les requêtes correspondant aux différents besoins utilisateurs :
   - SELECT pour lire et afficher les données,
   - INSERT pour ajouter de nouveaux enregistrements,
   - UPDATE pour modifier des données existantes,
   - DELETE pour supprimer des enregistrements.

Ainsi, la base de données est opérationnelle et peut être utilisée pour gérer efficacement les utilisateurs, événements et inscriptions de la plateforme.

2. Précisez les moyens utilisés :

Ordinateur, VS Code**, Google**

3. Avec qui avez-vous travaillé ?

Seule

4. Contexte

Nom de l'entreprise, organisme ou association ▸	**Centre de formation La Plateforme**
Chantier, atelier, service ▸	Atelier
Période d'exercice ▸	**Du** 29/07/25 **au** 29/07/25

Exemple n°2 ▸ *Développer des composants d'accès aux données SQL et NoSQL*
Développer des composants métier coté serveur

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

Description du projet : Conception du cœur du système d'une application web de gestion de tournois e-sport, sans se préoccuper de l'interface utilisateur.

1. Création du projet

La mise en place d'un projet PHP nécessite l'installation et le démarrage d'un serveur local (par exemple WAMP ou XAMPP). Pour ce projet, j'ai choisi d'utiliser **WAMP**.

Une fois le serveur démarré, j'ai créé un dossier nommé projet-php dans le répertoire /wamp64/www. Ce dossier contient l'ensemble des fichiers du projet.

Comme l'application doit reposer sur une base de données, j'ai créé une base esport via **phpMyAdmin** (code SQL fourni dans les consignes).

- La structure (tables, relations et clés étrangères) a été définie à l'aide de requêtes **DDL**.
- Les données de test ont été insérées grâce à des requêtes **DML** .

```sql
DROP DATABASE IF EXISTS esports;
CREATE DATABASE esports;
USE esports;

-- Table des utilisateurs
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) NOT NULL UNIQUE,
    email VARCHAR(100) NOT NULL UNIQUE,
    password_hash VARCHAR(255) NOT NULL,
    role ENUM('player', 'organizer', 'admin') NOT NULL DEFAULT 'player',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
)ENGINE=InnoDB;

-- Table des équipes
CREATE TABLE teams (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL UNIQUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
)ENGINE=InnoDB;

-- Table de liaison entre utilisateurs et équipes
CREATE TABLE team_members (
    id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT NOT NULL,
    team_id INT NOT NULL,
    role_in_team ENUM('member', 'captain') NOT NULL DEFAULT 'member',
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE,
    FOREIGN KEY (team_id) REFERENCES teams(id) ON DELETE CASCADE
)ENGINE=InnoDB;

-- Table des tournois
CREATE TABLE tournaments (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(150) NOT NULL,
    game VARCHAR(100) NOT NULL,
    description TEXT,
    start_date DATE NOT NULL,
    end_date DATE NOT NULL,
    organizer_id INT NOT NULL,
    FOREIGN KEY (organizer_id) REFERENCES users(id) ON DELETE CASCADE
)ENGINE=InnoDB;
```

```sql
-- Table d'inscription des équipes à un tournoi
CREATE TABLE registrations (
    id INT AUTO_INCREMENT PRIMARY KEY,
    team_id INT NOT NULL,
    tournament_id INT NOT NULL,
    registration_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    UNIQUE (team_id, tournament_id),
    FOREIGN KEY (team_id) REFERENCES teams(id) ON DELETE CASCADE,
    FOREIGN KEY (tournament_id) REFERENCES tournaments(id) ON DELETE CASCADE
)ENGINE=InnoDB;
```

```sql
-- Données de test : utilisateurs
INSERT INTO users (username, email, password_hash, role) VALUES
('admin_master', 'admin@esports.com', 'hashed_pwd_1', 'admin'),
('orga_jane', 'jane@orga.com', 'hashed_pwd_2', 'organizer'),
('player_tom', 'tom@players.com', 'hashed_pwd_3', 'player'),
('player_ana', 'ana@players.com', 'hashed_pwd_4', 'player'),
('player_max', 'max@players.com', 'hashed_pwd_5', 'player');

-- Données de test : équipes
INSERT INTO teams (name) VALUES
('Red Dragons'),
('Blue Phoenix'),
('Cyber Ninjas');

-- Données de test : membres d'équipes
INSERT INTO team_members (user_id, team_id, role_in_team) VALUES
(3, 1, 'captain'), -- Tom dans Red Dragons
(4, 1, 'member'),  -- Ana dans Red Dragons
(5, 2, 'captain'), -- Max dans Blue Phoenix

-- Données de test : tournois
INSERT INTO tournaments (name, game, description, start_date, end_date, organizer_id) VALUES
('Spring Smash Cup', 'Super Smash Bros', 'Tournoi printanier avec lots à gagner.', '2025-07-10', '2025-07-12', 2),
('League Ultimate', 'League of Legends', 'Tournoi 5v5 ouvert à tous.', '2025-08-01', '2025-08-03', 2);

-- Données de test : inscriptions
INSERT INTO registrations (team_id, tournament_id) VALUES
(1, 1),
(2, 1);
```

2. Initialisation

Tous les fichiers .php créés doivent contenir la structure de base d'un document HTML :

```
<!DOCTYPE html>
<html lang="fr">
<head>
  <!-- Métadonnées -->
</head>
<body>
  <!-- Contenu -->
</body>
</html>
```

Pour intégrer du code PHP à l'intérieur de ces fichiers, j'utilise les balises :

- <?php ... ?> pour écrire du code PHP classique
- <?= ... ?> pour afficher directement le résultat d'une expression (équivalent à echo).

Ensuite, il est nécessaire d'établir la connexion entre la base de données MySQL et l'application. Pour cela, j'ai créé un fichier db.php qui contient l'initialisation de l'objet **PDO** à l'aide d'un bloc try...catch. **PDO** permet une connexion sécurisée et portable entre PHP et MySQL

Cet objet $pdo peut ensuite être importé et réutilisé dans les autres fichiers afin d'exécuter les différentes requêtes SQL.

```php
<?php
$host = 'localhost';
$dbname = 'esports';
$username = 'dodo';
$password = '83210';

try {
    $pdo = new PDO(
        "mysql:host=$host;dbname=$dbname;charset=utf8",
        $username,
        $password,
        [PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION]
    );
} catch (PDOException $th) {
    die("Erreur de connexion à la base : " . $th->getMessage())
}
?>
```

3. US-1 : Création de compte

User Story : *En tant que nouveau participant, je veux pouvoir créer un compte avec email et mot de passe, afin de pouvoir rejoindre des équipes et participer à la plateforme.*

Mise en place du formulaire (HTML)

La fonctionnalité d'inscription est implémentée dans le fichier inscription.php, qui contient un formulaire d'inscription et des boutons de navigation.

- Le formulaire est codé en **HTML**, avec quatre champs input de type text et password.
- L'attribut action="#" signifie que le formulaire soumet les données à la même page (inscription.php).
- La méthode est POST, car il s'agit de données sensibles qui ne doivent pas apparaître dans l'URL.
- Un bouton "Déjà inscrit ?" permet à l'utilisateur de se rediriger vers la page de connexion s'il possède déjà un compte.

```html
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>E-SPORT GESTION : Inscription</title>

    <link rel="stylesheet" href="css/style.css">
</head>
<body>
<h1>Inscription</h1>

<form action="#" method="POST" id="form-inscription">
    <div>
        <label for="username">Nom d'utilisateur</label>
        <input type="text" id="username" name="username">
    </div>

    <div>
        <label for="email">E-mail</label>
        <input type="email" id="email" name="email">
    </div>

    <div>
        <label for="password">Mot de passe</label>
        <input type="password" id="password" name="password">
    </div>

    <div>
        <label for="password2">Confirmez votre mot de passe</label>
        <input type="password" id="password2" name="password2">
    </div>

    <input type="submit" value="S'inscrire" name="inscription">
</form>

<div id="options-inscription">
    <p>Déjà inscrit·e ? </p>
    <a href="connexion.php">Connectez-vous ici !</a>
</div>
```

Traitement côté serveur (PHP)

La page commence par session_start() afin d'activer la gestion des sessions. Grâce à if (isset(...)), on peut vérifier si une session existe déjà. Si c'est le cas, l'utilisateur est redirigé vers le hub de l'application, puisque la création d'un nouveau compte n'a pas lieu d'être.

```php
<?php
session_start();

if(isset($_SESSION['email'])){
    header('Location: hub.php');
    exit;
}
?>
```

Lorsque le formulaire est soumis (isset($_POST['inscription'])), les données sont récupérées et stockées dans des variables. Avant l'insertion en base, une série de **vérifications** est effectuée :

- Tous les champs sont remplis.
- Les deux mots de passe saisis sont identiques.
- L'adresse e-mail est valide.
- Le nom d'utilisateur et le mot de passe respectent une longueur minimale.
- L'adresse e-mail et le nom d'utilisateur sont uniques

```php
<?php
if(isset($_POST['inscription'])){
    $user_name = $_POST['username'];
    $email = $_POST['email'];
    $password = $_POST['password'];
    $password2 = $_POST['password2'];

    if(empty($email) || empty($password) || empty($password2) || empty($user_name)){
        echo("<p id='alert'>Tous les champs doivent être remplis :(</p>");
        exit;
    }

    if($password !== $password2){
        echo("<p id='alert'>La confirmation de votre mot de passe a échoué :(</p>");
        exit;
    }

    if(!filter_var($email, FILTER_VALIDATE_EMAIL)){
        echo("<p id='alert'>Votre adresse e-mail est incorrect :(</p>");
        exit;
    }

    if(strlen($user_name) < 4){
        echo("<p id='alert'>Votre nom d'utilisateur doit faire plus de 4 caractères :(</p>");
        exit;
    }

    if(strlen($password) < 8){
        echo("<p id='alert'>Votre mot de passe doit faire plus de 8 caractères :(</p>");
        exit;
    }
```

```php
require_once('db.php');

// Vérification si l'email est unique
$stmt_verif_email = $pdo->prepare('SELECT * FROM users WHERE email = :email');
$stmt_verif_email->execute(array(
    'email' => $email,
));

$verif_email = $stmt_verif_email->fetchColumn();

// Vérification si le nom d'utilisateur est unique
$stmt_verif_username = $pdo->prepare('SELECT * FROM users WHERE username = :username');
$stmt_verif_username->execute(array(
    'username' => $user_name,
));

$verif_username = $stmt_verif_username->fetchColumn();

if($verif_email){
    echo("<p id='alert'>Votre adresse e-mail est déjà liée à un compte existant :(</p>");
    exit;
}else if($verif_username){
    echo("<p id='alert'>Votre nom d'utilisateur est déjà utilisé :(</p>");
    exit;
}else{
```

Sécurité et insertion en base

Une fois les contrôles validés :

- Le mot de passe est **haché** avec password_hash(), car il ne faut jamais stocker de mots de passe en clair.
- La requête d'insertion (INSERT INTO) est préparée avec $pdo->prepare() et exécutée via execute(), afin de sécuriser la requête contre les injections SQL.

```php
    // Insertion
    $password_hash = password_hash($password, PASSWORD_BCRYPT);

    $stmt_insert = $pdo->prepare('INSERT INTO users(username, email, password_hash) VALUES (:username, :email, :password_hash)');

    $stmt_insert->execute(array(
        'username' => $user_name,
        'email' => $email,
        'password_hash' => $password_hash,
    ));

    $_SESSION['email'] = $email; //stockage email dans la session
    header("Location: hub.php");
    exit;
```

Création de session

Enfin, une session est créée en y stockant l'adresse e-mail de l'utilisateur. Celui-ci est ensuite redirigé vers le hub de l'application grâce à header("Location: ...").

```php
$_SESSION['email'] = $email; //stockage email dans la session
header("Location: hub.php");
exit;
```

4. US-2 : Connexion

User Story : *En tant qu'utilisateur enregistré, je veux pouvoir me connecter avec mon email et mot de passe, afin d'accéder à mes fonctionnalités personnalisées.*

Mise en place du formulaire (HTML)

La page connexion.php contient :

- Un formulaire permettant à l'utilisateur de saisir son email et son mot de passe
- Un bouton de navigation permettant de rejoindre la page d'inscription si l'utilisateur n'a pas encore de compte

```html
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>E-SPORT GESTION : Connexion</title>

    <link rel="stylesheet" href="css/style.css">
</head>
<body>

<h1>Connexion</h1>

<form action="#" method="POST" id="form-connexion">
    <div>
        <label for="email">E-mail</label>
        <input type="email" id="email" name="email">
    </div>

    <div>
        <label for="password">Mot de passe</label>
        <input type="password" id="password" name="password">
    </div>

    <input type="submit" value="Se connecter" name="connexion">
</form>

<div id="options-inscription">
    <p>Pas encore de compte ?</p>
    <a href="inscription.php">Inscrivez-vous ici !</a>
</div>
```

Traitement côté serveur (PHP)

Avant d'afficher la page, une vérification est effectuée : si une session est déjà active, l'utilisateur est redirigé vers le hub, car la connexion n'est plus nécessaire.

```php
<?php
session_start();

if(isset($_SESSION['email'])){
    header('Location: hub.php');
    exit;
}
?>
```

Lorsque le formulaire est soumis (isset($_POST['connexion'])), les données sont récupérées et stockées dans des variables.

Une série de vérifications est ensuite effectuée :

- Tous les champs sont remplis.
- L'adresse e-mail existe dans la base (via une requête SQL SELECT).
- Le mot de passe fourni correspond bien au mot de passe stocké en base. Pour cela, la fonction password_verify($motdepasse, $mdp_hash) est utilisée afin de comparer le mot de passe saisi avec la version hachée enregistrée. Cette fonction assure une gestion sécurisée des mots de passe, en respectant les bonnes pratiques de sécurité applicative.

```php
<?php
if(isset($_POST['connexion'])){
    $email = $_POST['email'];
    $password = $_POST['password'];

    if(empty($email) || empty($password)){
        echo("<p id='alert'>Tous les champs doivent être remplis :(</p>");
        exit;
    }

    require_once('db.php');

    $stmt = $pdo->prepare('SELECT password_hash FROM users WHERE email = :email');

    $stmt->execute(array(
        'email' => $email
    ));

    $user = $stmt->fetch(PDO::FETCH_ASSOC);
```

Création de la session

Si toutes les vérifications sont validées :

- Une session est créée, stockant l'adresse e-mail de l'utilisateur
- Celui-ci est redirigé vers le hub de l'application grâce à header("Location: ...")

```
    if($user !== false && password_verify($password, $user['password_hash'])){
        //stockage dans la session
        $_SESSION['email'] = $email;

        header("Location: hub.php");
        exit;
    }else{
        echo("<p id='alert'>Mot de passe et/ou indentifiant incorrect :(</p>");
    }
}
?>
```

5. US-3 : Déconnexion

User Story : *En tant qu'utilisateur connecté, je veux pouvoir me déconnecter de mon compte, afin de sécuriser mon accès*.

Mise en place (hub.php)

La page d'accueil (hub.php) comporte un bouton "Déconnexion" qui redirige l'utilisateur vers la page deconnexion.php.

```
<a href="deconnexion.php" class="button red">Déconnexion</a>
```

Page de déconnexion (deconnexion.php)

HTML : La page propose deux options : confirmer ou annuler la déconnexion.

```php
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>E-SPORT GESTION : Déconnexion</title>
    <link rel="stylesheet" href="css/style.css">
</head>
<body>
    <h1>Déconnexion</h1>
    <p id="p-logout">Êtes-vous sûr·e de vouloir vous déconnecter, <?=$user['username']?>?</p>

    <div id="div-button">
        <a href="index.php?deconnexion=yes" class="button red">OUI, Se déconnecter</a>
        <a href="hub.php" class="button">NON, Rester connecté·e</a>
    </div>

</body>
</html>
```

PHP :

- Si l'utilisateur choisit *Annuler*, il est redirigé vers hub.php.
- Si l'utilisateur confirme, l'URL inclut ?deconnexion=yes, ce qui déclenche le traitement de la déconnexion.

Suppression de la session (index.php)

Dans index.php, un contrôle est effectué avec isset($_GET ['deconnexion']).

Si le paramètre deconnexion est présent, la session active est détruite. L'utilisateur est alors redirigé vers la page d'accueil (index.php).

```php
<?php
if(isset($_GET['deconnexion'])){
    session_start();
    session_unset();     // Supprime les variables de la session
    session_destroy();   // Détruit la session

    $message = "Vous avez bien été déconnecté·e";
}else{
    session_start();
}
```

Cette gestion simple et sécurisée des sessions garantit que les utilisateurs peuvent fermer leur accès proprement, renforçant ainsi la sécurité de l'application.

6. US-4 : Modifier mon profil

User Story : *En tant qu'utilisateur, je veux pouvoir modifier mes informations personnelles, afin de garder mon profil à jour*.

Mise en place (**hub.php**)

La page d'accueil (hub.php) comporte un bouton "Modifier mes informations" qui redirige l'utilisateur vers la page modifier_compte.php.

Formulaire de modification - HTML (modifier_compte.php)

La page contient un formulaire permettant de mettre à jour le nom d'utilisateur et l'adresse e-mail. Deux boutons d'enregistrement permettent de valider les modifications.

```
<div>
    <label for="username">Nom d'utilisateur</label>
    <input type="text" id="username" name="username" value='<?=$user['username']?>'>
    <input type="submit" value="Mettre à jour" name="modifier-username">
</div>
```

```
<div>
    <label for="email">E-mail</label>
    <input type="email" id="email" name="email" value='<?=$user['email']?>'>
    <input type="submit" value="Mettre à jour" name="modifier-email">
</div>
```

Traitement côté serveur - PHP (modifier_compte.php)

Lorsqu'un formulaire est soumis (isset($_POST['modifier-username'])), les données saisies sont stockées dans des variables.

Avant toute mise à jour, une série de vérifications est effectuée à l'aide d'une requête SQL SELECT préparée avec $pdo->prepare() :

- La nouvelle donnée n'existe pas déjà en base
- Elle respecte la longueur minimale acceptée

```php
<?php
if(isset($_POST['modifier-username'])){
    $user_name = $_POST['username'];

    $stmt_username = $pdo->prepare('SELECT * FROM users WHERE username = :username');
    $stmt_username->execute(array(
        'username' => $user_name
    ));

    $verif_username = $stmt_username->fetchColumn();

    if($verif_username){
        echo("<p id='alert'>Votre nom d'utilisateur est déjà utilisé :(</p>");
    }else if(strlen($user_name) < 4){
        echo("<p id='alert'>Votre nom d'utilisateur doit faire plus de 4 caractères :(</p>");
    }else{
        $stmt_insert_username = $pdo->prepare('UPDATE users SET username = :username WHERE id = :id');

        $stmt_insert_username->execute(array(
            'username' => $user_name,
            'id' => $user['id']
        ));

        echo("<p id='alert'>Votre nom d'utilisateur a bien été modifié !</p>");
    }
}
```

```php
<?php
if(isset($_POST['modifier-email'])){
    $email = $_POST['email'];

    $stmt_email = $pdo->prepare('SELECT * FROM users WHERE email = :email');
    $stmt_email->execute(array(
        'email' => $email
    ));

    $verif_email = $stmt_email->fetchColumn();

    if($verif_email){
        echo("<p id='alert'>Votre adresse mail est déjà utilisé par un autre compte :(</p>");
    }else if(!filter_var($email, FILTER_VALIDATE_EMAIL)){
        echo("<p id='alert'>Votre adresse e-mail est incorrect :(</p>");
    }else{
        $stmt_insert_email = $pdo->prepare('UPDATE users SET email = :email WHERE id = :id');

        $stmt_insert_email->execute(array(
            'email' => $email,
            'id' => $user['id']
        ));

        echo("<p id='alert'>Votre adresse mail a bien été modifiée</p>");
    }
}
?>
```

Une fois ces vérifications passées, une requête UPDATE est préparée puis exécutée avec execute(), ce qui permet de mettre à jour la base de données.

7. US-5 : Créer une équipe

User Story : *En tant que joueur, je veux pouvoir créer ma propre équipe, afin de participer à des compétitions avec mes coéquipiers.*

Mise en place (hub.php / equipes.php / creer_equipe.php)

Depuis la page d'accueil (hub.php), l'utilisateur peut accéder à la page equipes.php. Cette page permet aux utilisateurs ayant un rôle ***player*** d'accéder au bouton "Créer une nouvelle équipe". Ce bouton redirige vers creer_equipe.php, où se trouve le formulaire de création d'équipe.

Formulaire de création - HTML (creer_equipe.php)

Le formulaire est simple et se compose de :

- Un champ input pour saisir le nom de l'équipe
- Un bouton d'envoi pour valider la création

```
<a href="equipes.php" class="button">Retour</a>
<h1>Création de ton équipe</h1>

<form action="" method="POST" id="form-team">
    <label for="nom">Nom d'équipe</label>
    <input type="text" id='nom' name="nom">

    <input type="submit" name="create">
</form>
```

Traitement côté serveur - PHP (creer_equipe.php)

Lors de la soumission du formulaire, le nom d'équipe est récupéré et stocké dans une variable. Une série de **tests de validation** est effectuée :

- Le champs n'est pas vide
- Le nom respecte la longueur minimale
- Le nom est unique en base

```php
if(isset($_POST['create'])){
    $nom = $_POST['nom'];

    if(empty($nom)){
        exit("<p id='alert'>Veuillez donner un nom à votre équipe</p>");
    }

    if(strlen($nom) < 3 || strlen($nom) > 25){
        exit("<p id='alert'>Votre nom d'équipe doit faire entre 3 et 25 caractères</p>");
    }

    // Vérification si le nom est unique

    $stmt_verif = $pdo->prepare('SELECT * FROM teams WHERE name = :nom');
    $stmt_verif->execute(array(
        'nom' => $nom
    ));

    $verif = $stmt_verif->fetchColumn();

    if($verif){
        exit("<p id='alert'>Ce nom d'équipe existe déja, veuillez en choisir un autre :(</p>");
    }
```

Si toutes les conditions sont respectées :

- Une requête SQL INSERT INTO est préparée et exécutée pour enregistrer la nouvelle équipe
- Une autre requête SQL INSERT INTO est ensuite exécutée pour ajouter les membres de l'équipe.

```php
$stmt_insert = $pdo->prepare('INSERT INTO teams(name) VALUES (:nom)');
$stmt_insert->execute(array(
    'nom' => $nom
));

$last_id = $pdo->lastInsertId(); // récupération de l'id de la team créée

// INSERT

$stmt_insert_captain = $pdo->prepare('INSERT INTO team_members(user_id, team_id, role_in_team) VALUES (:user_id, :team_id, :cap)');
$stmt_insert_captain->execute(array(
    'user_id' => $user['id'],
    'team_id' => $last_id,
    'cap' => 'captain'
));

// Vérification de l'insert

if($stmt_insert->rowCount() > 1){
    echo"<p id='alert'>Une erreur s'est produite :( Veuillez réessayer.</p>";
}else{
    echo"<p id='alert'>Votre équipe a été créée avec succès !</p>";
}
```

8. US-6 : Rejoindre une équipe

User Story : *En tant que joueur, je veux pouvoir rejoindre une équipe existante, afin de jouer en groupe.*

Mise en place (hub.php / equipes.php)

Depuis la page d'accueil (hub.php), l'utilisateur peut accéder à la page equipes.php, qui affiche la liste des équipes existantes.

Selon la relation entre l'utilisateur et l'équipe (non-membre, membre ou capitaine), différentes options sont proposées. Si le joueur n'appartient pas encore à l'équipe, un bouton "Rejoindre l'équipe" s'affiche. Ce bouton redirige vers la page rejoindre_equipe.php?id={$team['id']}, l'ID de l'équipe étant transmis dans l'URL.

```php
if($user['role'] === 'player'){

    if ($role === 'captain') {
        echo "<a href='gestion_equipe.php?id={$team['id']}'>Gérer l'équipe</a>";
    } else if (!$role) {
        echo "<a href='rejoindre_equipe.php?id={$team['id']}'>Rejoindre l'équipe</a>";
    }
}else if($user['role'] === 'admin'){
    echo "<a href='supprimer_equipe.php?id={$team['id']}'>Supprimer l'équipe</a>";
}
```

Formulaire de confirmation (rejoindre_equipe.php)

La page affiche un formulaire de confirmation pour rejoindre l'équipe. Un récapitulatif des informations de l'équipe est affiché grâce à une requête SQL SELECT. Le bouton "Rejoindre" confirme l'action et redirige l'utilisateur vers equipes.php?id=<?=$id_team?>&join=yes.

```php
<body>
    <a href="equipes.php" class="button">Retour</a>
    <h1>Rejoindre <?=$team['name']?> ?</h1>
    <p id="welcome">Êtes-vous sûr·e de vouloir rejoindre cette équipe ?</p>

    <div id='team'>
        <h2><?=$team['name']?></h2>
        <p>Date de création : <?=$team['created_at']?></p>
        <p>Nombre de membre : <?=$team['count']?></p>

        <div>
            <?php
            $stmt_equipe = $pdo->prepare('SELECT role_in_team, username FROM team_members AS tm LEFT JOIN users ON user_id = users.id WHERE tm.team_id = :id');
            $stmt_equipe->execute(array(
                'id' => $id_team
            ));

            $membres = $stmt_equipe->fetchAll(PDO::FETCH_ASSOC);

            foreach($membres as $membre) :
            ?>
                <div>
                    <h3><?=$membre['username']?></h3>
                    <p><?=$membre['role_in_team'] === "captain" ? "Capitaine" : "Membre" ?></p>
                </div>
            <?php
            endforeach
            ?>
        </div>
    </div>
    <a href="equipes.php?id=<?=$id_team?>&join=yes" class="button" style="margin : 15px auto">Rejoindre</a>
</body>
```

Traitement côté serveur (equipes.php)

Lors du retour sur equipes.php, l'URL est analysée.Si le paramètre join=yes est présent, une requête SQL INSERT INTO team_members est exécutée afin d'ajouter l'utilisateur à l'équipe correspondante. En cas de succès, un message de confirmation s'affiche pour informer l'utilisateur qu'il a bien rejoint l'équipe.

```php
if($join == "yes"){ //Rejoindre une équipe
    $stmt_join = $pdo->prepare('INSERT INTO team_members(user_id, team_id) VALUES (:user, :team)');
    $stmt_join->execute(array(
        'user' => $user['id'],
        'team' => $_GET['id']
    ));

    if($stmt_join->rowCount() > 1){
        echo"<p id='alert'>Une erreur s'est produite :( Veuillez réessayer.</p>";
    }else{
        echo"<p id='alert'>Vous avez rejoint votre équipe avec succès !</p>";
    }
}
```

2. Précisez les moyens utilisés :

Google, Ordinateur, VS Code, phpMyAdmin

3. Avec qui avez-vous travaillé ?

Seule

4. Contexte

Nom de l'entreprise, organisme ou association ▸	**Centre de formation La Plateforme**
Chantier, atelier, service ▸	Atelier
Période d'exercice ▸	**Du** 11/09/25 **au** 13/09/25

Exemple n°3 ▸ ***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 :

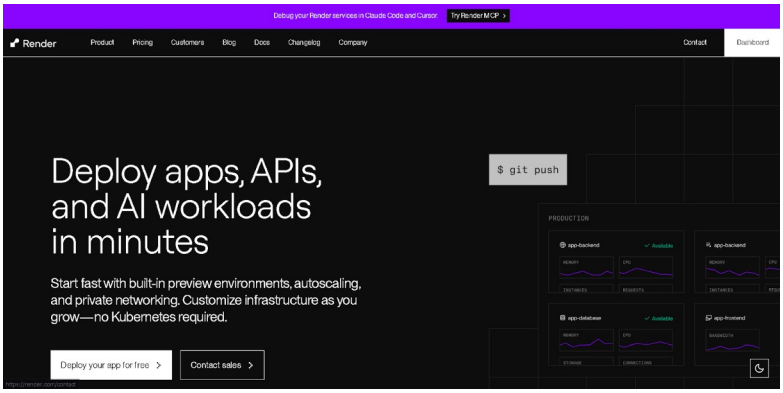
1. Déploiement du back-end avec Render

Pour le déploiement du back-end du projet, j'ai utilisé **Render**, un service d'hébergement disposant d'une formule gratuite adaptée aux petites applications.

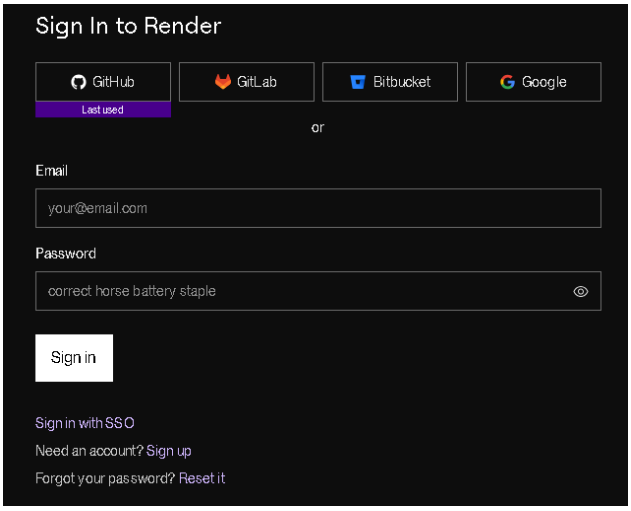
Étapes de configuration

Connexion et choix du dépôt

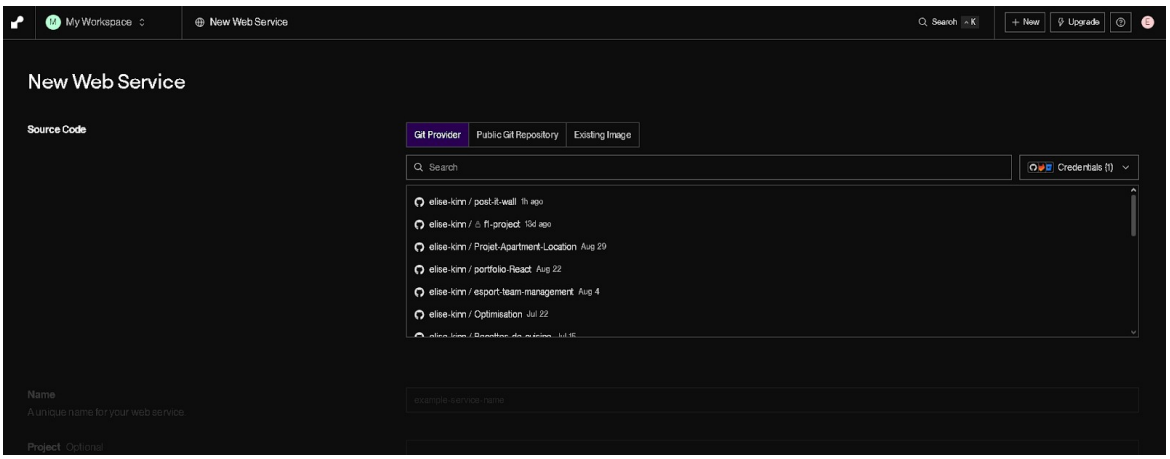
- Je me rends sur le site de Render et sélectionne *« Deploy your app for free »*.



- Je crée un compte en choisissant l'option *GitHub*, ce qui me permet de relier directement Render à mon dépôt.



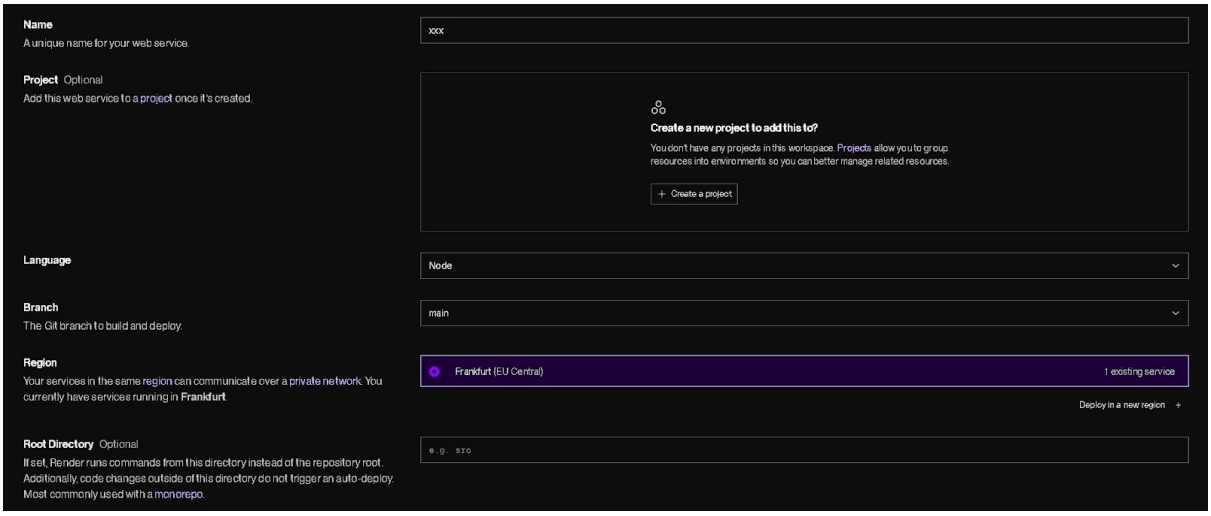
- En cliquant sur *« + New »*, je sélectionne le dépôt correspondant au projet à déployer.



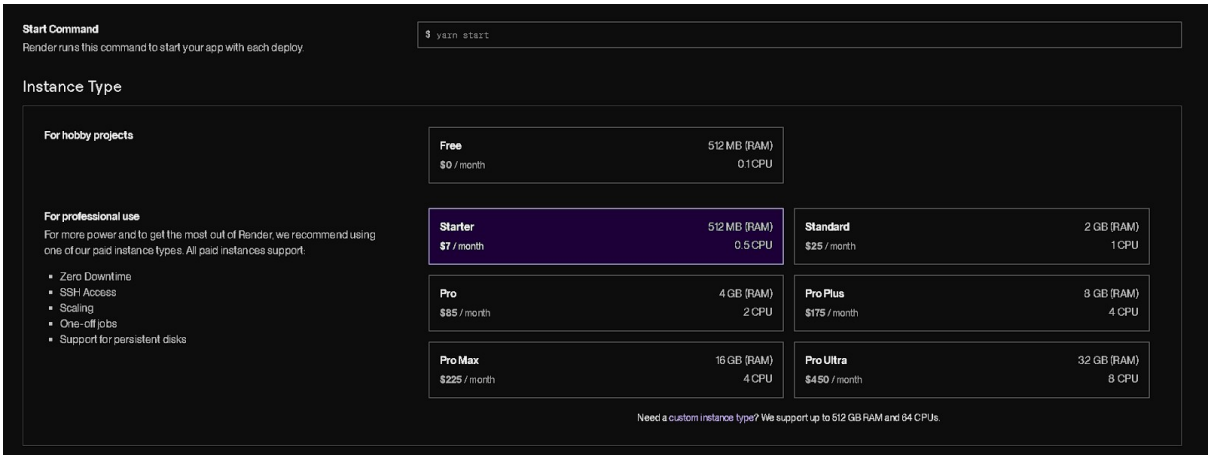
Paramètres de déploiement

Render me demande de renseigner plusieurs informations :

- **Name** : post-it wall
- **Language** : Node
- **Branch** : main
- **Region** : Frankfurt (EU Central)
- **Root directory** : back (le dossier contenant app.js et package.json)



- **Build Command** : `npm install`
- **Start Command** : `node ./app.js`
- **Instance Type** : Free



Variables d'environnement

Pour connecter l'application à la base de données et définir le port d'exécution, je configure les variables suivantes :

- DB_URI=mongodb+srv://<username>:<password>@cluster0.xxxxx.mongodb.net/nomDeLaBDD?retryWrites=true&w=majority&appName=Cluster0
- PORT=10000

Déploiement et résultat

Après validation des paramètres, Render lance le processus d'installation et de démarrage.

Une fois le déploiement terminé, une **URL publique** est générée (par exemple : *https://post-it-wall.onrender.com*), permettant d'accéder directement à l'API du back-end.

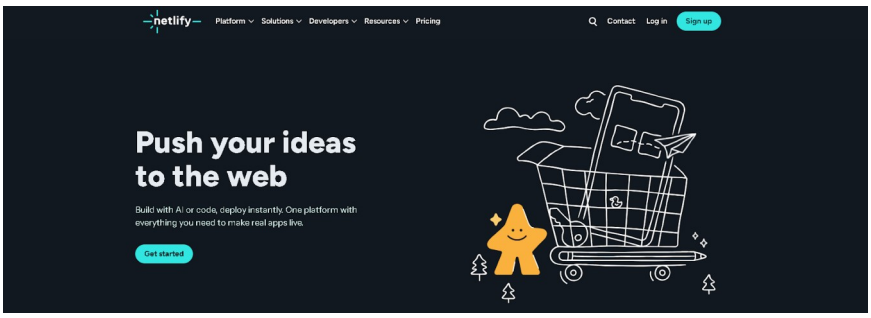
2. Déploiement du front-end avec Netlify

Pour le déploiement du front-end, j'ai utilisé **Netlify**, une plateforme spécialisée dans l'hébergement d'applications front-end avec un système d'intégration continue basé sur GitHub.

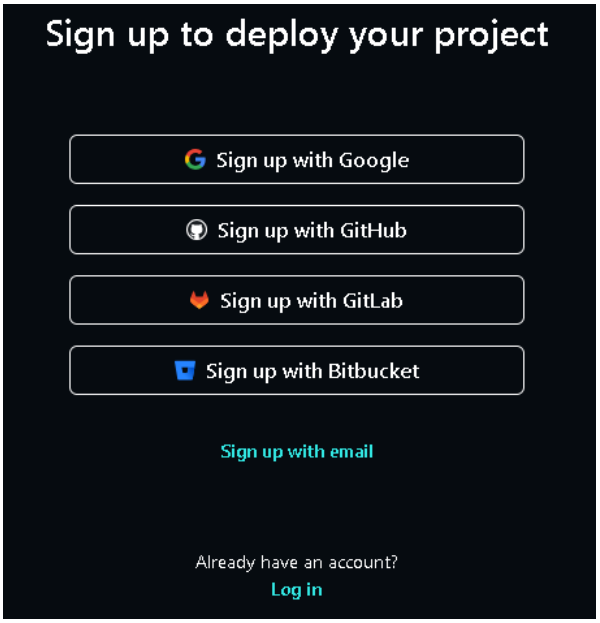
Étapes de configuration

Connexion et choix du dépôt

- Je me rends sur le site de Netlify et sélectionne “Get Started”.



- Je connecte mon compte GitHub à Netlify afin d’accéder directement à mes dépôts.

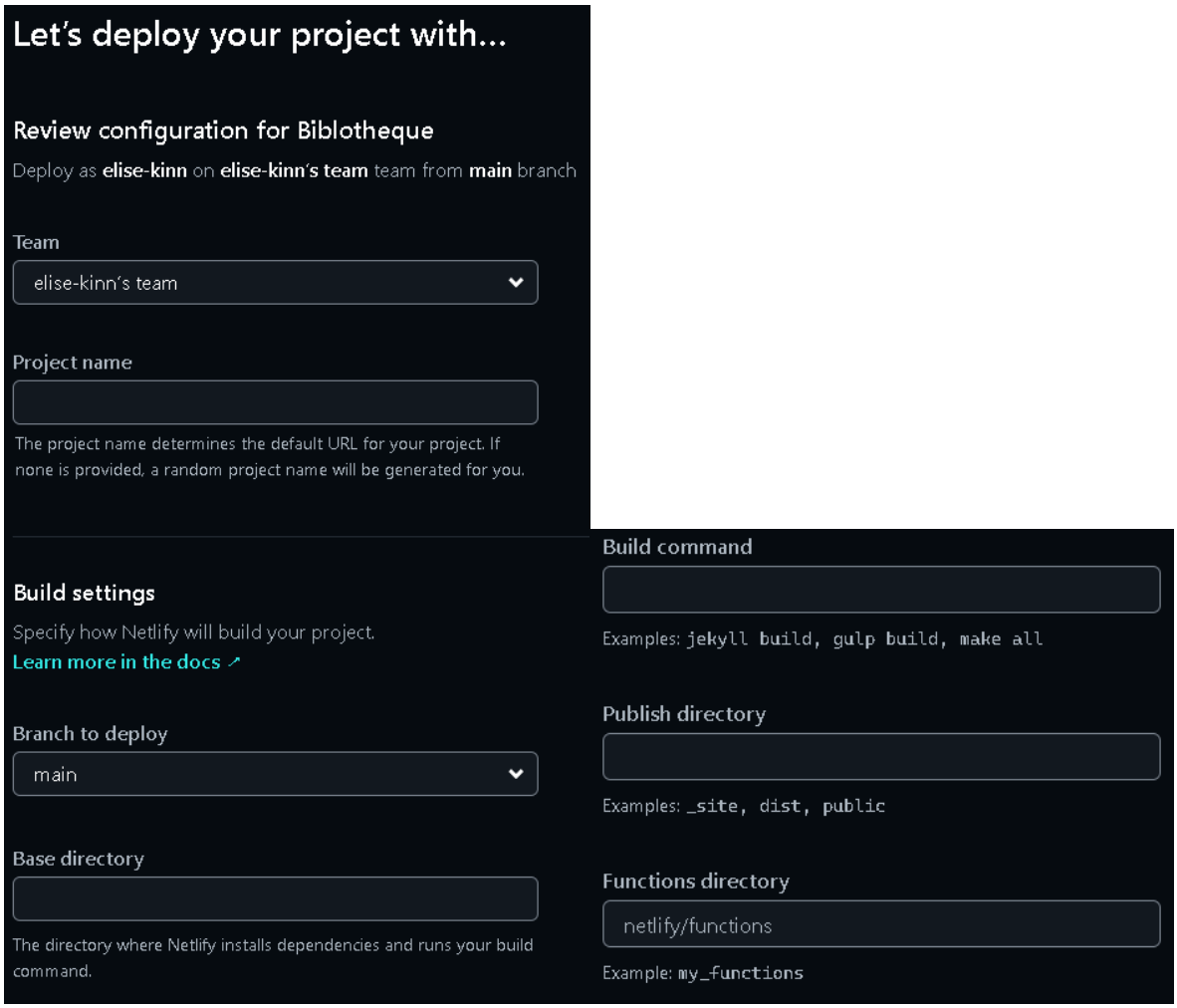


- Je sélectionne “Add New Project” → “Import an existing project” → “GitHub” et choisis le dépôt correspondant au projet à déployer.

Paramètres de déploiement

Lors de la configuration, Netlify demande plusieurs informations :

- **Project name** : post-it wall
- **Branch to deploy** : main
- **Base directory** : front
- **Build command** : npm run dev
- **Publish directory** :front/dist



Variables d'environnement

Pour permettre au front de communiquer avec le back déployé sur Render, je définis une variable d'environnement :

- VITE_API_URL=https://post-it-wall.onrender.com (URL de l'API du back déployé)

Déploiement et résultat

Une fois les paramètres enregistrés, Netlify lance automatiquement l'installation des dépendances et la génération du projet.

À l'issue du processus, une **URL publique** est créée (du type https://post-it-wall.netlify.app), permettant d'accéder directement au site en production.

2. Précisez les moyens utilisés :

Ordinateur, VS Code, Netlify, Render**, Google**

3. Avec qui avez-vous travaillé ?

Seule

4. Contexte

Chantier, atelier, service	Atelier
Période d'exercice	**Du** 25/09/25 **au** 27/09/25

Titres, diplômes, CQP, attestations de formation

(facultatif)

Intitulé	Autorité ou organisme	Date
Cliquez ici.	Cliquez ici pour taper du texte.	Cliquez ici pour sélectionner une date.
Cliquez ici.	Cliquez ici pour taper du texte.	Cliquez ici pour sélectionner une date.
Cliquez ici.	Cliquez ici pour taper du texte.	Cliquez ici pour sélectionner une date.
Cliquez ici.	Cliquez ici pour taper du texte.	Cliquez ici pour sélectionner une date.
Cliquez ici.	Cliquez ici pour taper du texte.	Cliquez ici pour sélectionner une date.
Cliquez ici.	Cliquez ici pour taper du texte.	Cliquez ici pour sélectionner une date.
Cliquez ici.	Cliquez ici pour taper du texte.	Cliquez ici pour sélectionner une date.
Cliquez ici.	Cliquez ici pour taper du texte.	Cliquez ici pour sélectionner une date.
Cliquez ici.	Cliquez ici pour taper du texte.	Cliquez ici pour sélectionner une date.
Cliquez ici.	Cliquez ici pour taper du texte.	Cliquez ici pour sélectionner une date.

Déclaration sur l'honneur

Je soussigné(e) [prénom et nom] *Elise LIAUTAUD*,

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

Fait à *BRIGNOLES* le *21/11/2025*

pour faire valoir ce que de droit.

Signature :

ELISE LIAUTAUD

Documents illustrant la pratique professionnelle

(facultatif)

Intitulé
Cliquez ici pour taper du texte.

ANNEXES

(Si le RC le prévoit)