

**Titre professionnel**

Développeur web et web mobile

Niveau III

**14/06/2022 – 22/03/2023**

**DOSSIER DE PROJET**

Réalisé par : Trong Hung VU

Stage : 09/01/2023 – 24/02/2023

**SOMMAIRE**

I. [Liste des compétences couvertes par le projet 3](#_Toc129897702)

[I.1. Développer la partie front-end de l'application 4](#_Toc129897704)

[I.2. Développer la partie back-end de l'application 5](#_Toc129897705)

[I.3. Produire les vidéos tutorielles avec la voix digitale naturelle 6](#_Toc129897706)

[I.4. Déploiement en ligne de la site web 6](#_Toc129897707)

II. [Résumé du projet 7](#_Toc129897709)

III. [Cahier des charges 9](#_Toc129897711)

[III.1. Projet pédagogique Blocnote 10](#_Toc129897712)

[III.2. Projet de paiement en ligne avec API Stripe 13](#_Toc129897713)

IV. [Spécifications techniques des projets 14](#_Toc129897715)

[IV.1. Choix technologiques 15](#_Toc129897716)

[IV.2. Sécurité 15](#_Toc129897717)

[IV.3. Méthodologie de gestion de projet 16](#_Toc129897718)

[IV.4. Validation des pages web 17](#_Toc129897719)

V. [Réalisations 18](#_Toc129897721)

[V.1. Projet pédagogique Blocnote 19](#_Toc129897722)

[V.2. Projet de paiement en ligne avec API Stripe 33](#_Toc129897723)

VI. [Jeu d'essai de la fonctionnalité la plus représentative 38](#_Toc129897725)

VII. [Veille sur les vulnérabilités de sécurité 42](#_Toc129897727)

VIII. [Description d'une situation ayant nécessité une recherche 45](#_Toc129897729)

IX. [Extrait du site anglophone et traduction 47](#_Toc129897731)

# I

# Liste des compétences couvertes par le

# projet

## Développer la partie front-end de l'application

* + 1. **Maquettage de l'application**

Le maquettage est une méthode de conception d'interface qui a permis de définir des interfaces visuelles conformes aux attentes et besoins. Suivant les demandes notées dans le cahier des charges, j’ai produit l’ensemble des pages de l’application pour l’affichage sur l’écran d’un desktop et d’un mobile. Pour réaliser le maquettage, j’ai utilisé l’outil **Figma**.

* + 1. **Créer une interface utilisateur web statique et adaptable**

Après le maquettage, l’intégration web des projets a été effectuée par **HTML** et **CSS**, permettant les pages web statiques. L’utilisation des CSS techniques telles que Flexbox et les media queries avec @media m’a permis l’interface totalement **Responsive** des pages en fonction de la taille des écrans. Les dimensions de l’écran sont concernées :

* la larguer (L) supérieure à 1440px pour les grands écrans comme la télévision,
* L de 1280px à 1440px pour l’ordinateur, et de 1025px à 1279px pour l’ordinateur de petite taille,
* L de 769px à 1024px pour les tablettes,
* L de 320px à 479px et de 480px à 768px pour les téléphones portables de toutes tailles
  + 1. **Créer une interface utilisateur web dynamique**

La plupart de mes pages web ont besoin de données, soit les notes de blog enregistrées dans la base de données MySQL soit le formulaire de paiement en ligne géré par API Stripe.

Pour pouvoir insérer les notes dans les pages, j’ai imprégné des codes de PHP (back-end) dans le code HTML du front-end. L’utilisation des boucles PHP m’a permis de créer automatiquement l’affichage HTML des notes.

Mais, lors de l’utilisation des templates Twig pour le projet Blocnote-Twig, les données des blog notes ont été récupérées par PHP et insérées au front-end à l’aide de la fonction **Render** de Twig.

Lors de l’utilisation de ReactJS, les pages HTML du front-end ont été créées et rendues dynamiquement à l’aide de Javascript. Comme Javascript fonctionne du côté front-end et les données de la base de données se trouvent du côté back-end, un API (Application Programming Interface) a été créé et exploité en utilisant la librairie **Axios** de Node.js.

Lors du projet de paiement en ligne avec API Stripe, le formulaire de Stripe a été créé et inséré dans ma page web à l’aide de **Javascript** dont Fetch API et les manipulations de DOM étaient importants. J’ai également utilisé le package **Vite** qui est un outil front-end JavaScript permettant d’améliorer la rapidité de développement en offrant un serveur de développement rapide et une compilation optimisée pour la production.

## Développer la partie back-end de l'application

* + 1. **Créer une base de données**

Le projet pédagogique Blocnote a besoin d’une base de données **MySQL** pour stocker toutes les notes de blog. Un simple modèle conceptuel des données (MCD) a été produit. J’ai utilisé l’outil **phpMyAdmin** pour créer et gérer la base de données.

* + 1. **Développer les composants d'accès aux données**

Pour le projet pédagogique, les manipulations de la base de données, comme Create Read Update Delete (CRUD), ont été codées avec l’objet **POO (PHP)**. Comme ça, j’ai pu sécuriser l’accès aux données en définissant les types de données entrés dans chaque champ de formulaire. De plus, l’utilisation des fonctions bindParam() ou bindValue() de l’objet PDO dans les requêtes SQL a permis d’éviter les attaques de type « injection SQL ».

Pour le projet Blocnote réalisé avec ReactJs, les requêtes HTTP en vue des actions CRUD avec la base de données ont été codées avec le framework **ExpressJS** en milieu Node.js. Les données entrantes des requêtes HTTP ont été récupérées à l’aide du module **body-parser** de Node.js. Ce module extrait entièrement la partie « body » d'un flux de la requête entrante et l'expose sur le paramètre « req.body ».

* + 1. **Développer la partie back-end de l’application en intégrant les recommandations de sécurité**

Lors des projets pédagogiques, la partie back-end a été développée sous **PHP natif** et sous **POO**. Comme ce projet est une démonstration qui vise différentes techniques de codage, le côté sécurité de l’application n’était pas bien apprécié. Pourtant, les recommandations de sécurité pour éviter des attaques de type « injection SQL » ont été prises en compte. Pour la version en ReactJS, sa partie back-end a été codée avec **ExpressJS**.

Lors du projet de paiement en ligne avec API Stripe, selon les demandes du maître de stage, la partie back-end de l’application a été codée avec PHP natif. J’ai également développé les fonctions de validation en PHP et les expressions régulières (RegEx) en vue de renforcer la sécurité des données du côté back-end.

## Produire les vidéos tutorielles avec la voix digitale naturelle

Après l’étude exhaustive des techniques Text-to-Speech dans l’objectif d’une voix robotique « naturelle » et gratuite, nous avons décidé d’utiliser le logiciel **NaturalReader**, en version ‘en ligne’, pour produire la voix digitale des vidéos tutorielles. J’ai pu utiliser gratuitement les voix Premium pendant 20 minutes par jour, la durée suffisante pour réaliser 1 ou 2 vidéos de 5 à 10 minutes par vidéo. De plus, j’ai pu utiliser les voix gratuites disponibles de manière illimitée pour faire des tests de l’audio, tels que la vitesse de la voix, la prononciation, la synchronisation avec la vidéo, avant l’enregistrement des vidéos.

La capture vidéo d’écran a été réalisée à l’aide du logiciel **OBS Studio** (Open Broadcaster Software)**.** OBS Studio est une solution gratuite et Open Source qui permet de diffuser du contenu sur différentes plateformes de streaming.

Après l’enregistrement de la vidéo avec la voix digitale, la vidéo a été éditée en utilisant le logiciel de montage vidéo **Shotcut**. Shotcut est un éditeur vidéo multiplateforme Open Source et gratuit qui accepte un large éventail de formats audio, vidéo et d'image pour l'édition ou la conversion.

## Déploiement en ligne de la site web

Pour pouvoir gérer à distance un compte serveur hébergé sur PlanetHoster, j’ai utilisé le logiciel **MobaXterm** qui est un émulateur de terminal Linux pour Windows. Le protocole de communication sécurisé SSH a été configuré.

Afin de déployer le projet pédagogique codé avec ReactJs et ExpressJS, j’ai configuré l’environnement de Node.js pour le projet en installant les dépendances (package.json). Comme le Node.js du serveur utilise les modules de CommonJS (2009), j’ai dû adapter mes modules de ECMAScript 6 (ES6, 2015).

Lors du déploiement du projet de paiement en ligne, j’ai installé Composer et le package stripe/stripe-php sur le serveur avant la mise en place mes fichiers de code.

La base de données MySQL a été créée, et gérée à l’aide de l’outil phpMyAdmin du serveur. Les données ont été migrées depuis Local en employant les fonctions Import/Export de cet outil.

# II

# Résumé du projet

**Résumé du projet**

Le stage de 7 semaines (du 09/01/2023 au 24/02/2023) a été réalisé à distance auprès de l’agence web AB Digital à BOULOGNE BILLANCOURT (<https://sitedigital.fr/>). Pendant le stage, 2 projets ont pu être réalisés, soit le projet pédagogique (multi-techno) et le projet de paiement en ligne avec API Stripe.

L’objectif du projet pédagogique est de produire les vidéos tutorielles de développement web en montrant la possibilité de coder un site web par multi techniques de programmation. Un site simple de blog, nommé Bloc-Note, a été proposé, qui demande toutefois l’interaction entre le front-end, le back-end et la base de données. La page d’accueil contient toutes les notes enregistrées dans la base de données MySQL. En cliquant sur une note, la page détaillant la note s’affiche. Une note peut être ajoutée à la base de données via la page de l’ajout de note qui contient un formulaire à remplir. Le site a été réalisé avec PHP natif, et puis une autre version avec PHP orienté objet (POO), et enfin une version avec ReactJS et ExpressJS. La version en PHP natif a été réalisée sans et puis avec le moteur de templates Twig. L’utilisation des voix digitales naturelles dans les vidéos tutorielles a été étudiée et appliquée. J’ai produit 5 vidéos pour la version en PHP natif (de 5 à 11 minutes par vidéo) avec l’aide du logiciel de capture vidéo d’écran OBS Studio et du logiciel de montage vidéo Shotcut. De plus, j’ai pu déployer en ligne la version codée avec ReactJS et ExpressJS, dont le site web est <https://apibnr.abdigitalbusiness.fr/>.

Sur le site web de l’agence, lors du paiement, les clients seront dirigés vers une page Checkout de Stripe avec la mise en page prédéfinie, et puis redirigés vers la page initiale après le paiement. Cette procédure est moins appréciée en termes de l’UX. L’objectif du projet est d’intégrer le paiement de Stripe dans la page de vente de l’agence. Pour pouvoir mettre en place l’API Stripe, l’API PaymentIntent de Stripe a été choisi, qui est à jour et permet la vérification de 3D Secure des paiements par carte bancaire alors que l’API Charges de Stripe ne la permet pas. Le Fetch API de Javascript a permis les requêtes HTTP entre le front-end et le back-end. Après le paiement, les informations nécessaires de la vente, telles que l’identification du paiement, le nom et l’émail du client, ont été enregistrées dans un fichier .csv qui sera servi pour envoyer la formation (en vidéos) au client. A la fin du stage, j’ai pu déployer mon travail en ligne : <https://stage.abdigitalbusiness.fr/> (site protégé par un mot de passe).

# III

# Cahier des charges

## Projet pédagogique Blocnote

* + 1. **Contexte et présentation du projet**

Un site web peut être codé par différentes techniques de programmation. Le langage HTML (HyperText Markup Language) ne permet que la partie front-end de l’application. Le langage du côté back-end comme PHP – Hypertext Preprocessor (et ses frameworks tels que Laravel, Symfony) permet de coder la partie back-end de l’application qui peut ensuite générer des codes HTML des pages web de la partie front-end. Les différentes pages HTML sont ainsi chargées après le traitement logique des données de la partie back-end, ce qui en résulte une Multi Page Application – **MPA**.

Cependant, une partie ou l’entier du contenu d’une page HTML peut être dynamiquement modifié grâce à Javascript, ce qui en résulte une autre page HTML sans le chargement de la page entière. Cette approche permet une Single Page Application – **SPA**. La SPA utilise les APIs développés par le développeur back-end afin de lire et afficher les données enregistrées dans la base de données.

Par rapport à une application MPA, une SPA présente les avantages intéressants tels que :

- la vitesse plus élevée de chargement de l’application,

- la structure plus claire où les parties de front-end et de back-end sont séparées,

- le meilleur UX (User Experience),

- la réutilisation possible des codes back-end,

mais également les désavantages tels que :

- le SEO (Search Engine Optimization) plus mauvais

- moins de sécurité. Une SPA subi plus d’attaques des « hackers ».

- sa dépendance entière de Javascript.

L’objectif du projet pédagogique est de produire les vidéos tutorielles de formation pour montrer aux débutants différentes techniques de codage d’un site web. L’approche de MPA et SPA a été envisagée en vue d’une meilleure compréhension de ces techniques.

Le site web choisi était un blog, nommé Bloc-Note. Il contient 3 pages :

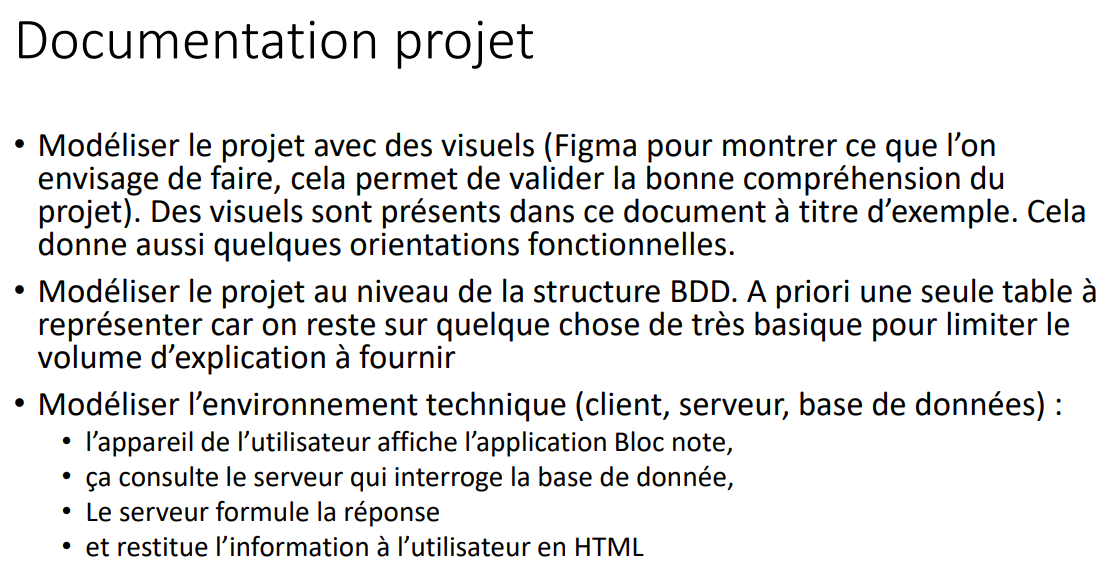
- La page d’accueil contient toutes les notes enregistrées dans la base de données.

- En cliquant sur une note, la page détaillant la note s’affiche.

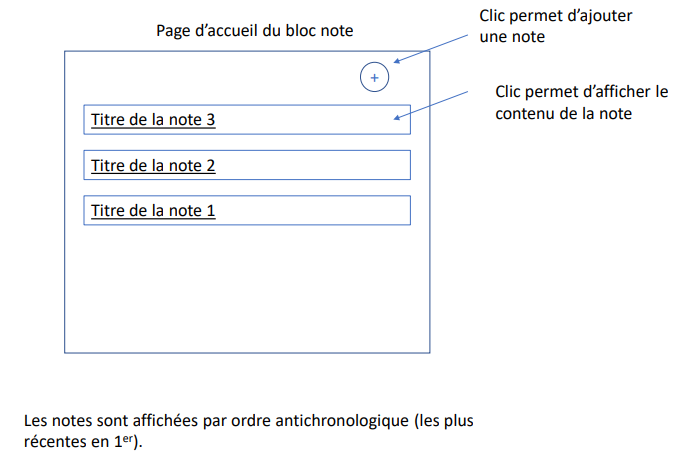
- Une note peut être ajoutée à la base de données via la page de l’ajout de note qui contient un formulaire à remplir.

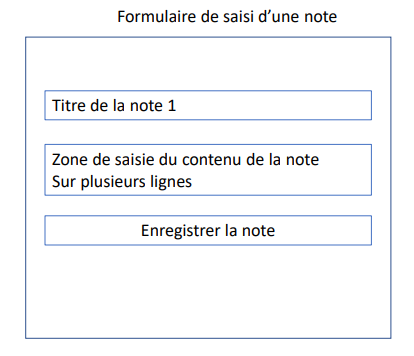
* + 1. **Expression des besoins**

Les besoins du projet m’ont été présentés par le maître de stage :

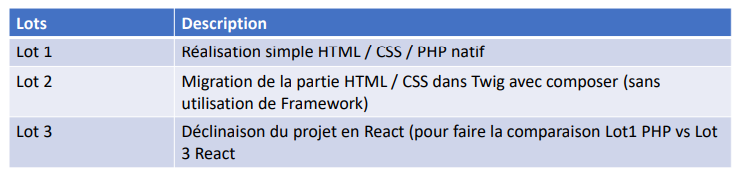


Il a proposé quelques idées de la maquette :





Les lots du projet étaient :



D’après la demande du maître de stage : « suivant la complexité et l’avancement on pourra limiter à 1 seule version => ce qui rend la digitalisation de la voix intéressante pour qu’une personne ensuite reprenne sans que la forme change etc) », les vidéos de formation ont dû utiliser la voix digitale « naturelle ».

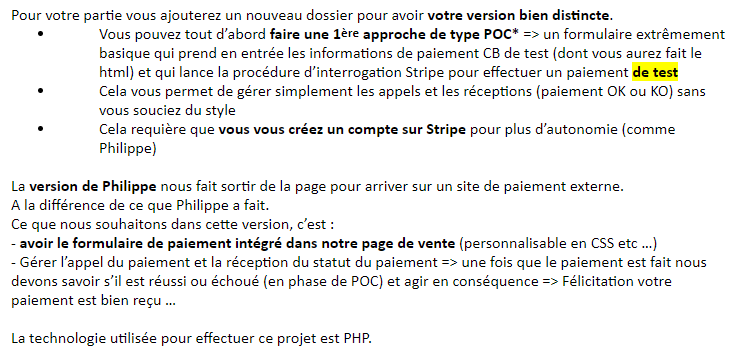
## Projet de paiement en ligne avec API Stripe

* + 1. **Contexte et présentation du projet**

Sur le site web de l’agence, lors du paiement en ligne, les clients seront dirigés vers une page Checkout de Stripe avec la mise en page prédéfinie, et puis redirigés vers la page initiale après le paiement. Cette procédure est moins appréciée en termes de l’UX. L’objectif du projet est d’intégrer le paiement de Stripe dans la page de vente. Lors de mon stage, j’ai intégré le paiement en ligne sur la page de vente créée par un autre stagiaire.

* + 1. **Expression des besoins**

Les besoins du projet m’ont été adressés :

****

**\*POC : Proof Of Concept**

# IV

# Spécifications techniques des projets

## Choix technologiques

D’après le cahier des charges ci-dessus, j’ai utilisé le langage **PHP** pour coder le back-end du projet Blocnote en version PHP. Les pages HTML du front-end ont été générées par les codes PHP.

Lors du projet Blocnote en version ReactJS, la partie front-end a été séparément développée avec **ReactJS** qui est une bibliothèque JavaScript libre et facilite la création d'application web monopage SPA. La large popularité actuelle de ReactJS s’explique par les hautes performances, la facilité d’utilisation et son évolutivité. Et la partie back-end du projet a été codée avec le framework **ExpressJS** de Node.js, qui m’a permis de produire les APIs afin de réaliser les actions CRUD avec la base de données.

La base de données était **MySQL**. Elle fait partie des logiciels de gestion de [base de données](https://fr.wikipedia.org/wiki/Base_de_donn%C3%A9es) les plus utilisés au monde, autant par le grand public (applications web principalement) que par des professionnels, en concurrence avec Oracle, PostgreSQL et Microsoft SQL Server. D’ailleurs, l’utilisation de MySQL m’a facilité le déploiement de mes applications web car le compte serveur hébergé sur PlanetHoster de l’agence web utilise également MySQL.

Lors du projet de paiement en ligne avec API Stripe, la partie back-end a été développée et sécurisée avec **PHP**. La base de données n’a pas été employée car nous ne stockons pas les informations des méthodes de paiement des clients pour la raison de sécurité.

## Sécurité

1. **Projet pédagogique**

Les mesures de la sécurité de l’application n’étaient pas parmi les objectifs du projet. Dans nos vidéos de formation, nous n’expliquons pas en détail aux débutants le côté sécurité de l’application. Cependant, les bonnes pratiques du codage ont été prises en compte, permettant la bonne sécurité de l’application.

* Injection SQL

Les injections SQL consistent à insérer dans un champ de formulaire le code SQL pour modifier le comportement de la requête SQL. Ainsi, un utilisateur mal intentionné peut faire des modifications pour obtenir plus de privilèges, ou même supprimer la base de données.

Pour éviter les injections SQL de ce projet, j’ai utilisé la fonction **htmlspecialchars**() pour nettoyer les saisis de l’utilisateur, et l’expression « placeholder » suivie par la fonction **bindParam**() pour protéger les requêtes SQL.

* Faille XSS (cross-site scripting)

La faille XSS consiste à écrire du code Javascript dans un champ de formulaire pour que, lorsque celui-ci est affiché sur la page après la validation du formulaire, puisse agir directement sur le site web, en provoquant des actions de l’utilisateur qui visite la page web.

Pour éviter la faille XSS, j’ai utilisé la fonction **htmlspecialchars**() de PHP qui converti des caractères spéciaux en entités HTML. Mais pour la version codée avec ReactJS, cette fonction n’existe pas en Javascript. Je l’ai donc défini en créant les fonctions EscapeHtml et DecodeHtml utilisant la fonction **replace**() de Javascript.

D’ailleurs, l’utilisation de **Twig** lors du lot 2 utilisant PHP et Twig a également permis d’éviter ce type de faille puisque Twig échappe automatiquement les valeurs des variables qu’elle affiche.

1. **Projet de paiement en ligne**

Comme ce projet sera présenté au grand public, la sécurité de l’application était importante. La clé secrète qui a été stockée du côté back-end en PHP n’est pas donc exposée aux utilisateurs. Les champs de formulaire ont été validés à l’aide des expressions régulières RegEx appliquées à la fois en Javascript du front-end et en PHP du back-end. De plus, j’ai créé les fonctions de validation en PHP pour sécuriser les requêtes HTTP.

## Méthodologie de gestion de projet

1. **Outil de stockage**

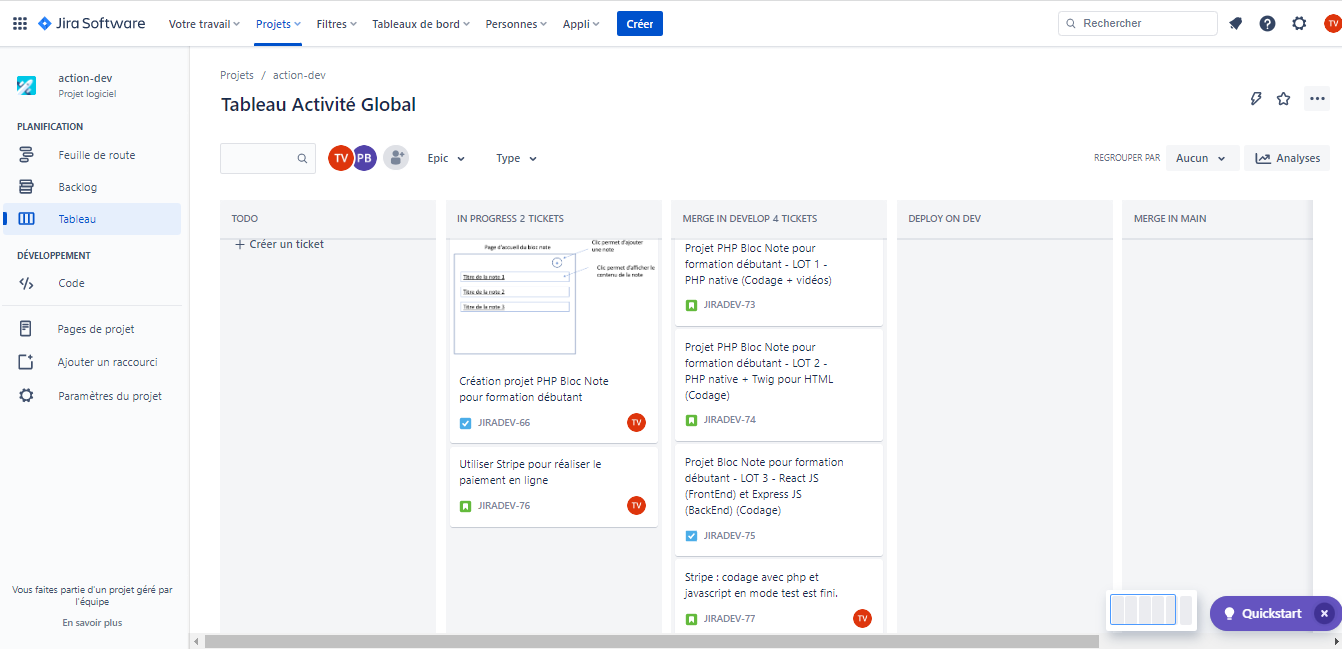
Tous les documents de stage, les images, les vidéos tutorielles de formations, les dossiers de l’administration, etc, sauf les codes source, ont été stockés sur Google Drive dans un dossier de partage avec le maître de stage.

1. **Outil de communication**

Les réunions à distance avec le maître de stage ont été réalisées à l’aide de l’outil Google Meet. Une chambre de réunion a été créée par le maître de stage. Cet outil nous a permis de partager l’écran pour montrer des codes, et le chat texte et vocal.

1. **Gestion de projet Jira**

La gestion des projets a été assurée à l’aide du logiciel Jira initié par le maître de stage. Les tâches ont été créées et gérées. Ainsi l’avancement des projets a été suivi avec facilité.



1. **Git et Gitlab**

Un dépôt en ligne sur Gitlab a été créé par le maître de stage. J’ai mis les codes source à jour dans ce dépôt sur ma branche Develop. Comme j’étais le seul développeur de ces projets, je n’ai pas eu de problèmes de conflit de code. Cependant l’utilisation de Git et Gitlab (ou Github) nous était indispensable, me permettant de stocker à jour mes codes et de récupérer des codes si besoin.

## Validation des pages web

Après l’intégration des pages web avec HTML et CSS, j’ai vérifié la qualité des pages à l’aide de l’outil Lighthouse qui évalue 4 critères de la page, soit la performance, l’accessibilité, les bonnes pratiques et le SEO. Cette évaluation m’a permis les corrections de code. De plus, le moteur de recherche Google "aime" les sites web de qualité. Plus la qualité est élevée, plus le site web est remarqué et sera au « top » des recherches.

D’ailleurs, j’ai également validé la structure des pages via le site du **W3C** (World Wide Web Consortium). Cela m’a permis non seulement de me conformer aux standards du web, mais également de repérer des erreurs de code.

# V

# Réalisations

## Projet pédagogique Blocnote

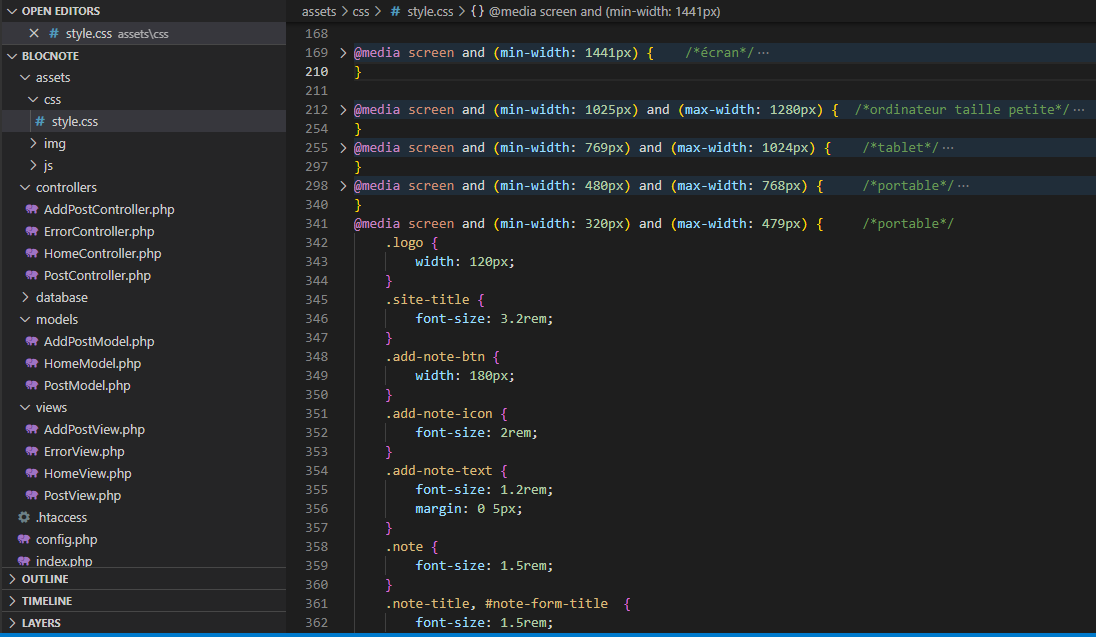
* + 1. **Version en PHP natif**
       1. Maquettage

Le projet Blocnote contient 3 pages. Basé sur le cahier des charges, j’ai réalisé les maquettes des pages pour l’affichage sur Desktop et Mobile à l’aide de l’outil Figma.

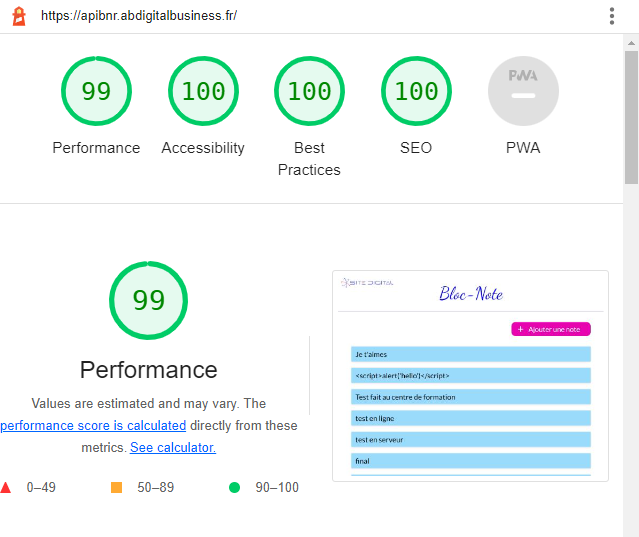


* + - 1. Intégration avec HTML et CSS

D’après les maquettes des pages, l’intégration avec HTML et CSS a été réalisée pour les 3 pages de l’affichage Desktop dont la largeur de l’écran est de 1281px à 1440px. L’utilisation de la technique Flexbox et des media queries de CSS m’a permis l’interface totalement **Responsive** des pages en fonction de la taille des écrans.



J’ai également vérifié les pages à l’aide de l’outil Lighthouse et du site **W3C.** Je présente les résultats obtenus de Lighthouse :

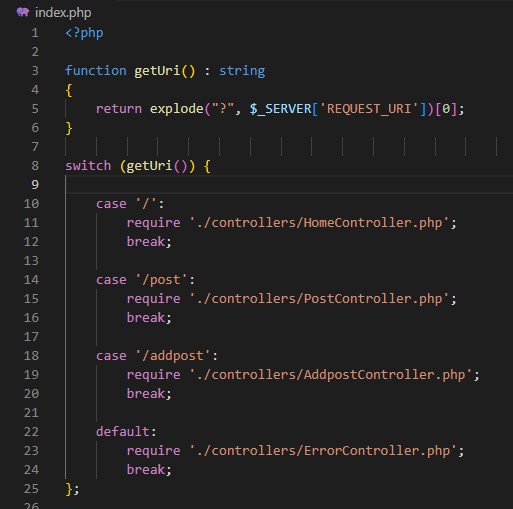


* + - 1. Architecture Modèle-Vue-Contrôleur (MVC)

Au niveau de l’organisation de code, j’ai employé l’architecture MVC, une des plus célèbres architectures dans le domaine de développement web. Elle m’a permis de bien organiser le code, comme montré dans l’image ci-dessus :

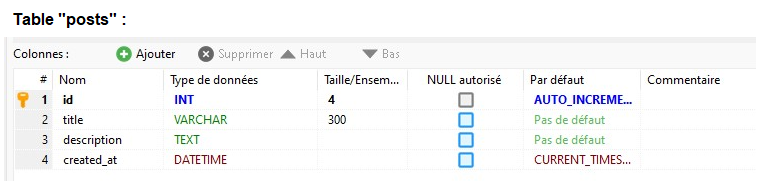
* Le contrôleur est la partie logique qui gère les échanges avec l'utilisateur. C'est en quelque sorte l'intermédiaire entre l'utilisateur, le modèle et la vue.
* Le modèle assure la gestion des données stockées dans la base de données et garantit leur intégrité.
* La vue est la partie qui se concentre sur l'affichage. Elle ne fait presque aucun calcul et se contente de récupérer des variables pour savoir ce qu'elle doit afficher.
  + - 1. Routeur

Comme j’ai eu plusieurs fichiers contrôleur, j’ai dû créer les routes de mon application. Pour ce faire, la technique de la redirection d’URL a été mise en œuvre, permettant de rediriger toutes les adresses d’URL vers un seul fichier de démarrage index.php à l’aide du fichier **.htaccess**. **Ainsi les routes ont été définies dans ce fichier** index.php. La fonction getUri() a permis de récupérer la valeur URI des adresses d’URL.



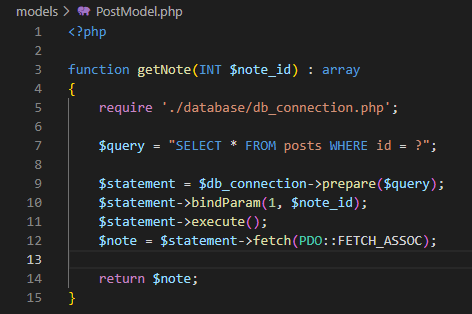
* + - 1. Base de données et son intégration

Utilisant l’outil phpMyAdmin, la base de données « blocnote » et sa seule table « posts » ont été créées :

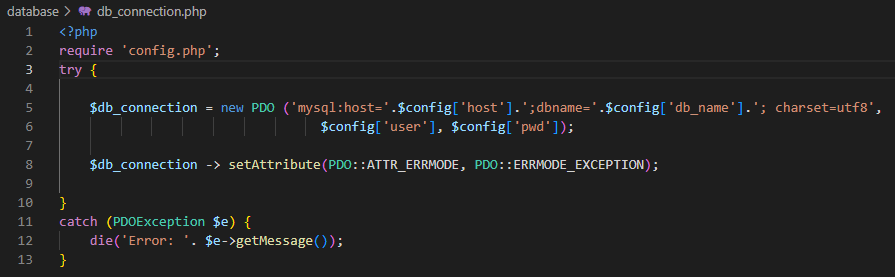


La récupération et l’insertion des données dans la base de données ont été réalisées en utilisant l’objet PDO de PHP dans les fichiers Modèle. Les requêtes SQL ont été préparés avec l’expression « placeholder ». Les variables de SQL ont été introduites et filtrées à l’aide de la fonction bindParam() pour éviter les injections SQL.

Je présente ici un exemple de la fonction getNote() dans le fichier PostModel.php afin de récupérer tous les détails de la note demandée.

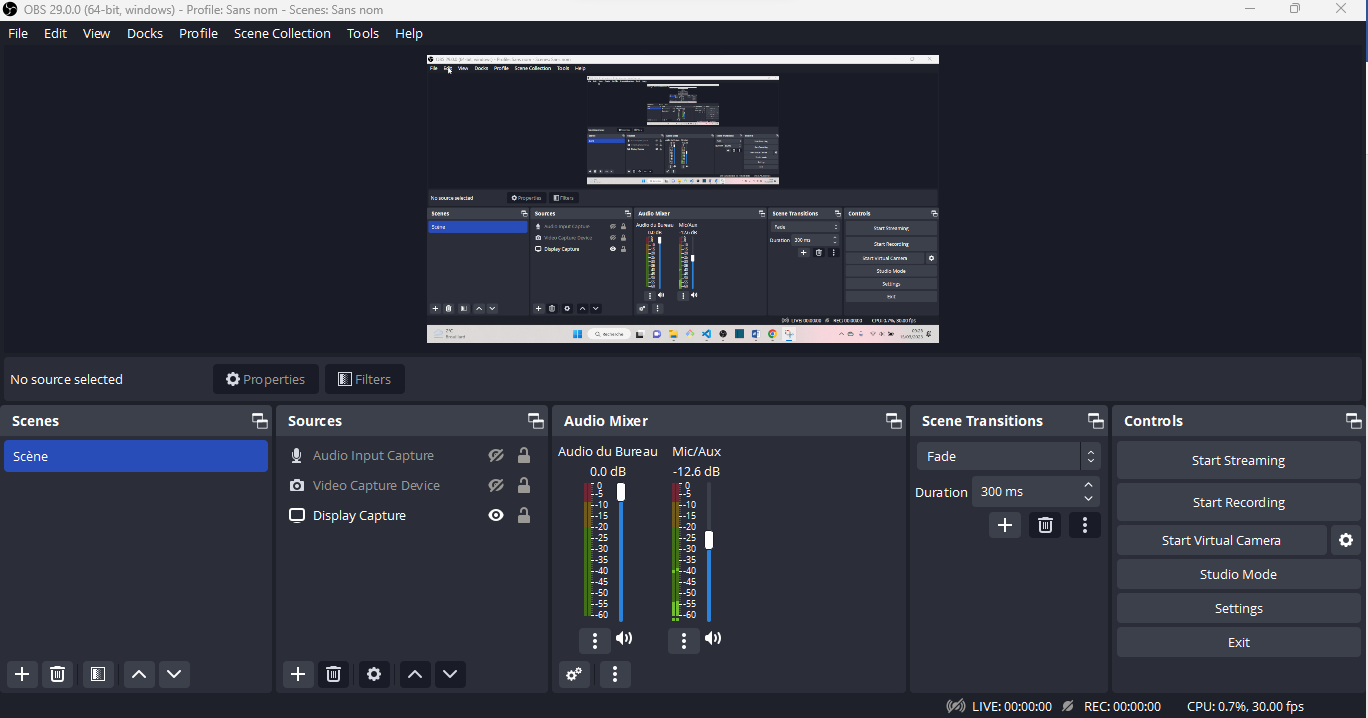


L’accès à la base de données a été établi sous la variable $db\_connection en utilisant l’objet PDO comme suit :



* + - 1. Production des vidéos tutorielles de la version en PHP natif

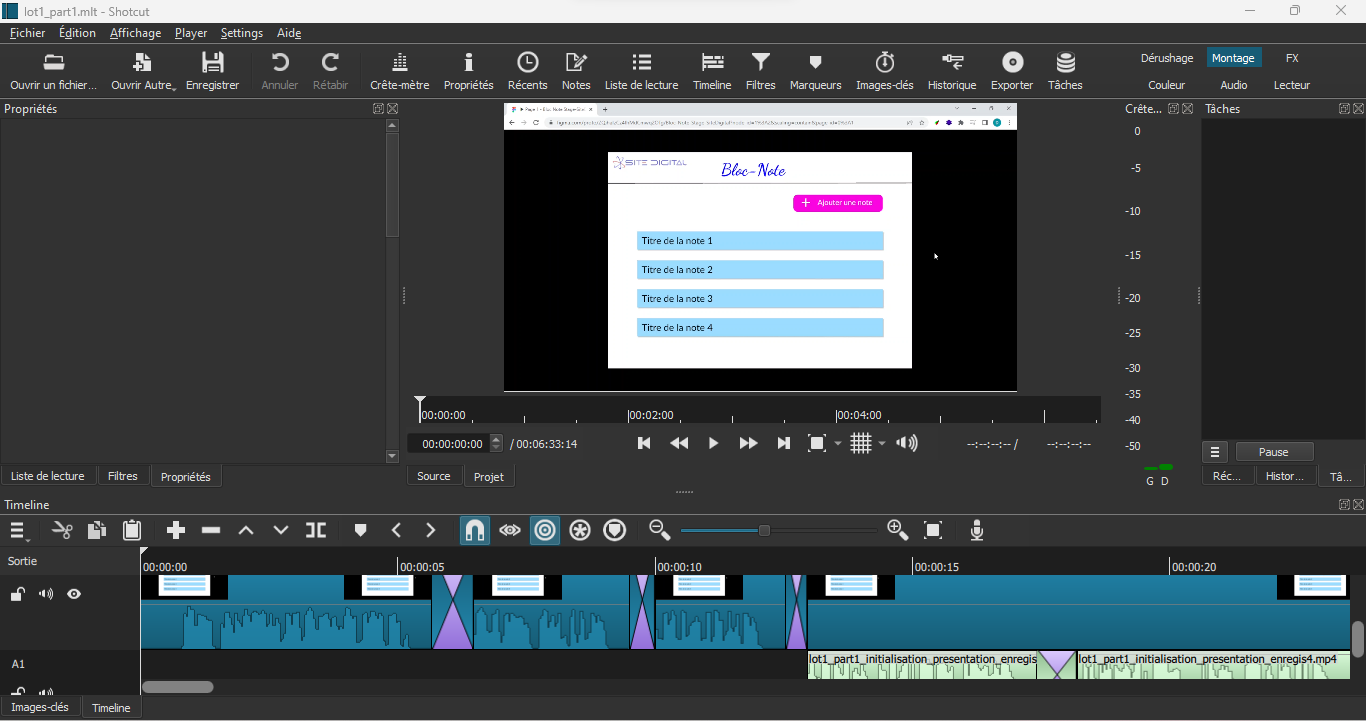
Tout d’abord, j’ai préparé le texte de la vidéo. Les tests ont été réalisés avec la voix gratuite du logiciel NaturalReader en ligne. Une fois tout est prêt, la vidéo de formation a été enregistrée à l’aide du logiciel OBS Studio dont les paramètres de la vidéo tels que la dimension, le « frame rate » fps, etc, ont été établis selon la vidéo standard de Youtube.



La voix Premium « naturelle » de Sophia a été choisie pour la vidéo de formation. Son utilisation est limitée à 20 minutes d’enregistrement par jour.

L’enregistrement de la vidéo était en parallèle avec l’émission de la voix digitale. La synchronisation entre la vidéo et l’audio était difficile, en particulier pour les vidéos de formation demandant plusieurs rédactions de code. J’ai donc dû faire plusieurs segments de l’enregistrement pour une vidéo de formation.

Ainsi l’utilisation du logiciel de montage vidéo Shotcut était indispensable pour corriger la synchronisation, enlever les images indésirées, cacher les taches, etc…



Finalement j’ai produit 5 vidéos tutorielles de formation de 5 à 11 minutes par vidéo pour cette version de code.

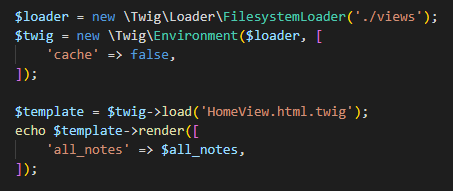
* + 1. **Version en PHP natif et Twig**

L’intégration du moteur de templates Twig dans le projet m’a demandé 2 modifications importantes :

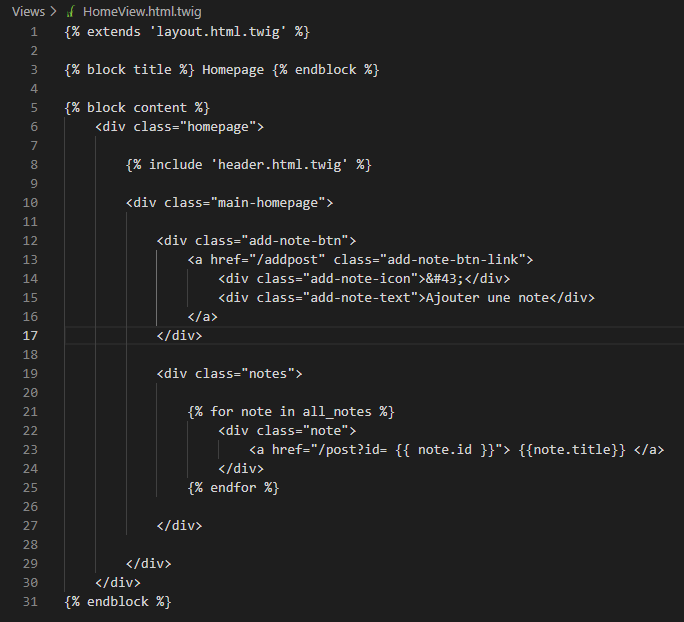
* Pour la partie back-end : L’installation du moteur de templates Twig comme une dépendance de l’application a été réalisée à l’aide de l’outil Composer en tapant la commande :

composer require twig/twig

Ensuite, j’ai défini les paramètres de Twig pour qu’il puisse charger les pages HTML de mon site web. Voici un exemple :



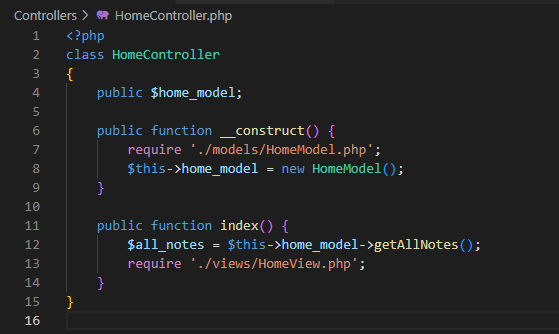
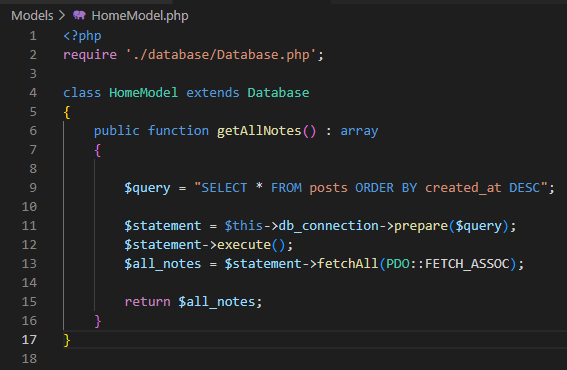
* Pour la partie front-end : Les pages HTML ont été ré-organisées. La partie identique des pages a été définie comme le template de base contenant des variables de contenu. Il s’agit du fichier layout.html.twig. En alimentant ces variables par les contenus correspondants, les templates des pages tels que HomeView.html.twig, PostView.html.twig, … ont été créés. Ainsi les pages HTML correspondantes ont été produites grâce à la fonction render() de Twig. Voici un exemple du template HomeView.html.twig :



* + 1. **Version en PHP Orienté Objet (POO)**

La partie front-end de la version en POO restait identique à celle de la version en PHP natif. Pourtant la partie back-end de l’application a été re-codée entièrement en appliquant les notions de Class de POO. Tous les fichiers Contrôleur, Modèle et l’accès à la base de données ont été définis sous une Class correspondante possédant ses propres propriétés et méthodes.

Je présente ici un exemple de la Class HomeController et de la Class HomeModel pour l’affichage de la page d’accueil :

* + 1. **Version en ReactJS et ExpressJS**
       1. Partie front-end avec ReactJS

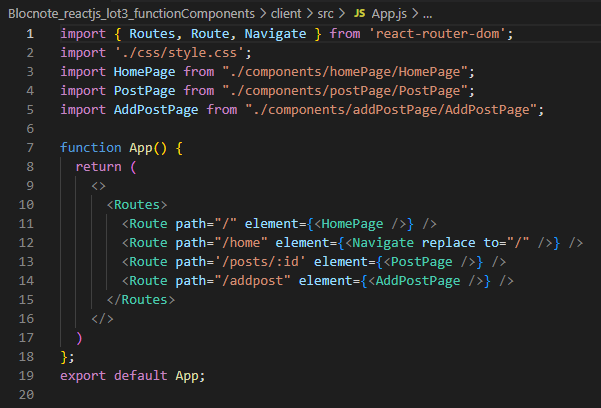
D’abord, j’ai installé la librairie ReactJS dans le sous-dossier « client » du projet par la commande :

npx create-react-app client

et ensuite les dépendances : le module react-router-dom et la librairie Axios à l’aide de l’outil **npm**.

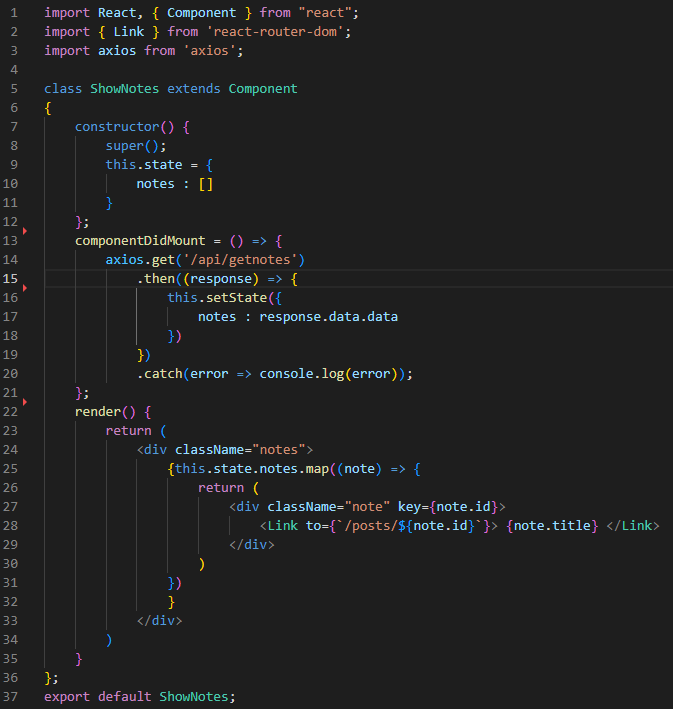
La partie front-end fonctionne à la porte http://localhost:3000/.

Le module react-router-dom m’a permis d’établir les routes de l’application :



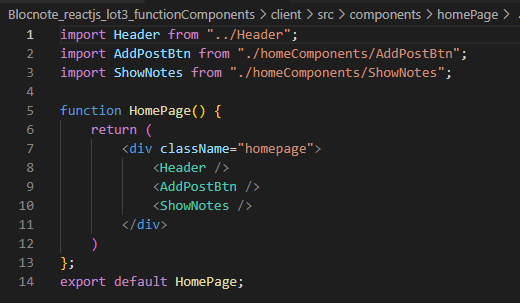
Comme présenté dans l’image ci-dessus, chaque page a été définie comme un component de l’application. Le component HomePage correspond à la page d’accueil, PostPage à la page de la note, et AddPostPage à la page de l’ajout de note.

Les components ont été écrits sous forme des fonctions depuis 2019 grâce à la nouvelle fonctionnalité **Hooks** de ReactJS. Les components avaient été codés avec la notion **Class**. Avec Hooks, la notion **Fonction** devient préférée par les développeurs par rapport à la notion Class pour coder les components. Par contre les anciens « class » components existent encore et Facebook n’envisage pas de les remplacer par les « fonction » components car Class présente encore ses avantages. Dans ce projet, j’ai codé les components avec la notion Fonction. Par contre, pour pouvoir consulter les anciens « class » components et ma curiosité, j’ai également codé les « class » components pour une version séparée. Voici un exemple du « class » component ShowNotes :

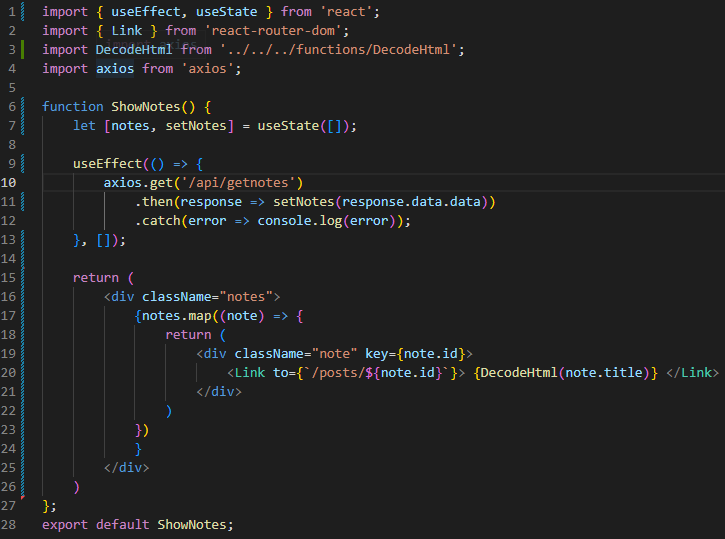


Pour un « fonction » component, par exemple HomePage :

* Chaque élément important de la page a été également défini comme un component, soit les components Header, AddPostBtn pour le bouton de l’ajout de note et ShowNotes, comme suit :

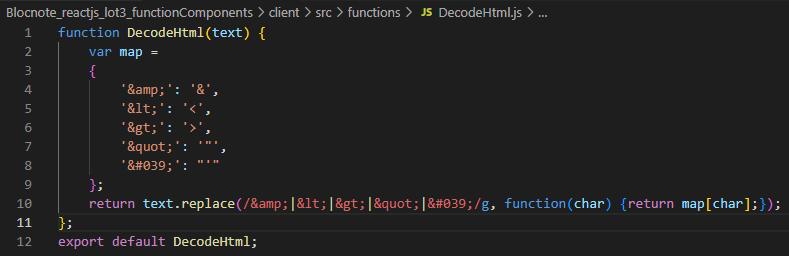


* Pour le component ShowNotes, les données des notes enregistrées dans la base de données ont été récupérées à l’aide de la librairie **Axios** et de l’API créé dans la partie back-end.



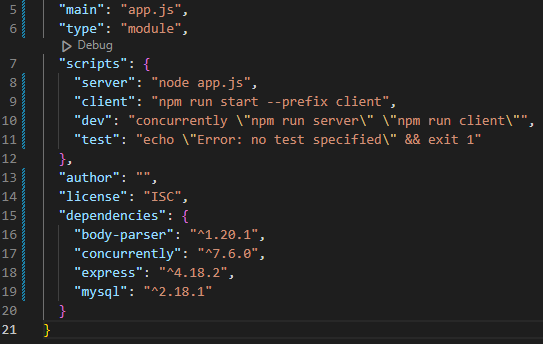
Comme la fonction htmlspecialchars() de PHP n’existe pas en version Javascript, les modules EscapeHtml (pour le component AddPostPage) et DecodeHtml (pour les components HomePage et PostPage) ont été créés afin de nettoyer les caractères spéciaux, permettant d’éviter les attaques de type « faille XSS ».





* + - 1. Partie back-end avec ExpressJS de Node.js

D’abord j’ai initié la partie back-end dans l’environnement de Node.js via la commande npm init, ce qui m’a ensuite permis d’installer les dépendances :

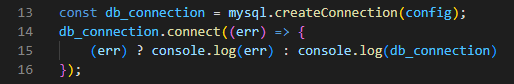


* Le module body-parser : permet mon API de récupérer entièrement les données envoyées par le front-end.
* Le module concurrently : permet de lancer le simultanément les serveurs back-end et front-end. Ce module a été utilisé pour faciliter le « développement » mais pas inclus dans la « production ».
* Le framework express (ExpressJS) : pour coder la partie back-end de l’application
* Le module mysql : permet de connecter à la base de données MySQL.

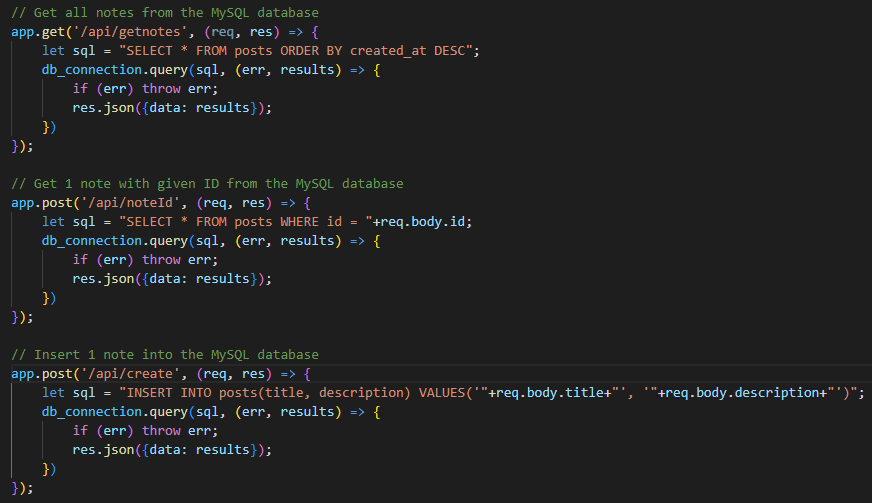
La partie back-end, codée dans le fichier app.js, fonctionne à la porte http://localhost:4000/.



L’accès à la base de données MySQL a été établi, et stocké dans la variable db\_connection :



Et j’ai défini un API en vue des requêtes HTTP avec la partie front-end :

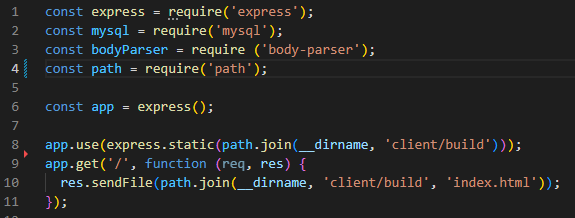


* + - 1. Déploiement de la version en ReactJS et ExpressJS

Le maître de stage m’a autorisé l’accès à son compte serveur sur l’hébergeur web Planethoster. J’ai utilisé le logiciel **MobaXterm** qui m’a permis de travailler à distance avec le compte serveur. Le protocole de communication sécurisé SSH a été configuré.

J’ai rencontré le problème de la compatibilité de version Node.js. Le serveur utilise les modules de CommonJS (2009) pour charger le fichier de démarrage app.js alors que j’ai utilisé les modules en ECMAScript 6 (ES6, 2015) pour mon code. J’ai dû modifier mon code en remplaçant import/export par **require/module.exports**.

Ensuite, j’ai produit la version « production » de la partie front-end en ReactJS dans le dossier « build » par la commande npm run build. Cette version de la partie front-end a été ainsi lancée grâce au fichier de démarrage app.js de la partie back-end :



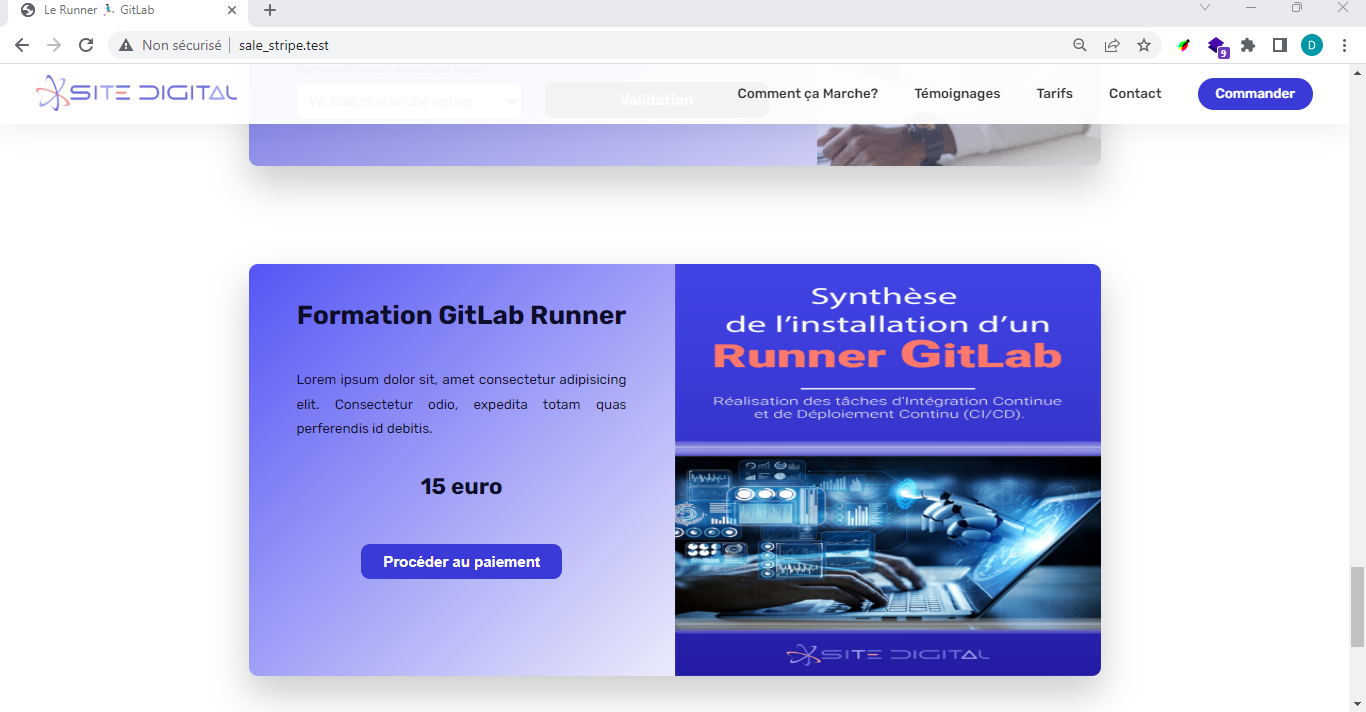
Le site web en ligne est : <https://apibnr.abdigitalbusiness.fr/>.

## Projet de paiement en ligne avec API Stripe

* + 1. **Partie front-end avec HTML, CSS et Javascript**

Dans un premier temps, j’ai intégré le paiement en ligne dans la page de vente réalisée par un stagiaire. Il m’a fallu créer une section HTML de paiement.

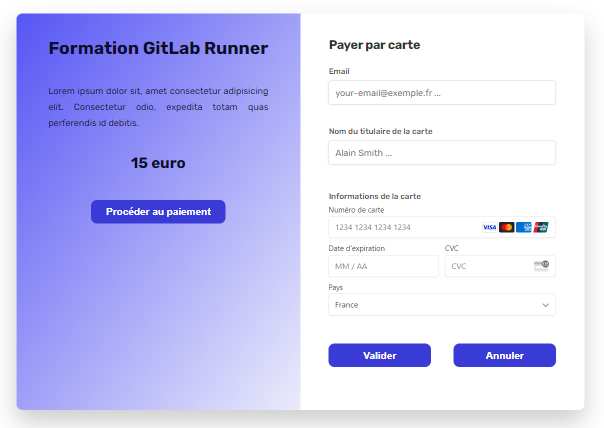
Comme la section était simple, je n’ai pas fait le maquettage. Je l’ai directement codé avec HTML et CSS. La Responsive de la page a été également assurée en utilisant les techniques Flexbox et media queries de CSS.



En cliquant sur le bouton « Procéder au paiement », l’image à droite est remplacée par le formulaire de paiement sans rafraîchissement de la page grâce à Javascript.

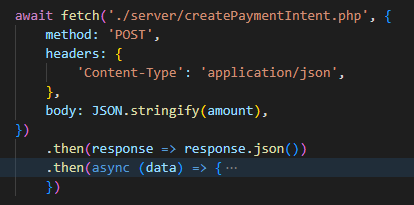
J’ai adopté l’API PaymentIntent de Stripe pour réaliser le paiement en ligne. L’API PaymentIntent est à jour et permet la vérification 3D Secure que toutes les cartes bancaires européennes requièrent. Cependant, j’ai également essayé l’API Charges qui est plus simple mais ne m’a pas permis la vérification 3D Secure.

Le formulaire « Informations de la carte » dans l’image ci-dessous est donc généré et validé par l’API PaymentIntent de Stripe :



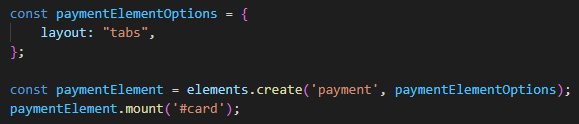
Selon l’API PaymentIntent de Stripe, qu’est-ce qui se passe lorsque le client clique sur le bouton « Procéder au paiement » ?

1. Le montant 15 euro de la vente est envoyé vers le back-end par une requête HTTP en utilisant la méthode fetch API :



1. Le back-end va retourner une intention de paiement de Stripe avec la clé secrète. Cette clé est simplement l’identité de l’intention de paiement. Ce n’est pas important si l’utilisateur cherche à le savoir via l’outil Inspecter du navigateur. La clé permet de créer un élément de paiement de Stripe, qui est intégré dans la page sous le formulaire de paiement présenté sur l’image au-dessus.



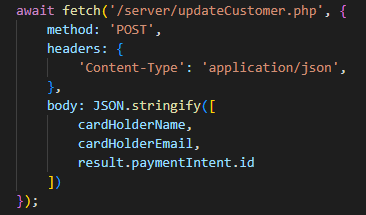


Les informations de la carte bancaire des clients, telles que le numéro de la carte bancaire, la date d’expiration et le code CVC, sont collectées, vérifiées et validées uniquement par Stripe. Ainsi, les informations erronées sont alertées immédiatement via les messages d’erreur.

1. Après avoir rempli les bonnes informations de la carte bancaire, en cliquant sur le bouton « Valider », le paiement sera validé par API Stripe auprès de la banque du client, et le montant sera affiché dans le compte Stripe en ligne du vendeur.



1. Si le paiement est réussi, le nom et l’émail du client seront envoyés vers le back-end pour le stockage dans le fichier .csv. Cette information sera utilisée pour envoyer la formation (sous forme des vidéos) au client.

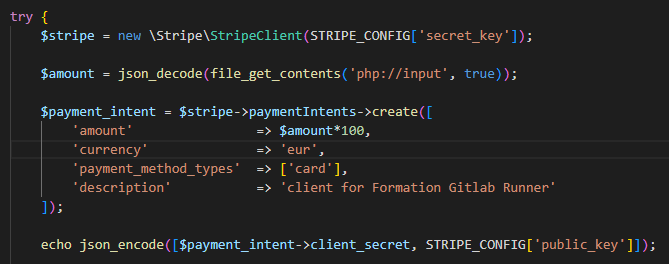


* + 1. **Partie back-end avec PHP**

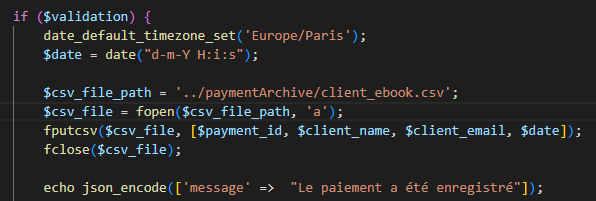
L’API Stripe a été installé comme une dépendance à l’aide de Composer :

composer require stripe/stripe-php

Reçu le montant de paiement venant du front-end, j’ai instancié une intention de paiement contenant la clé secrète de paiement. Cette clé sera envoyée au front-end pour créer un élément de paiement de Stripe.

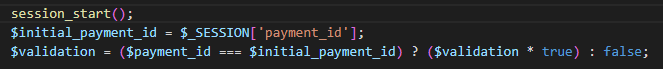


Ensuite, après la validation d’un paiement, les données telles que le nom, l’émail du client et l’identité du paiement sont stockées dans le fichier .csv.

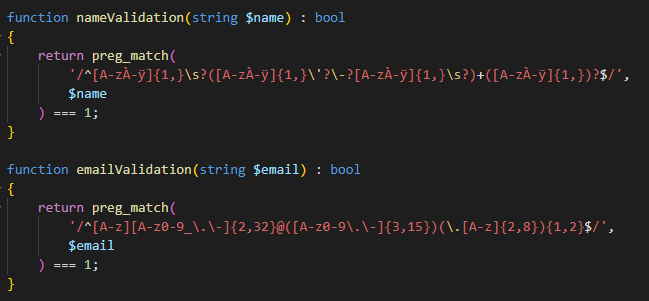


Pour éviter des mauvaises manipulations de ces données, j’ai créé les fonctions de validation comme suit :

* La vérification de l’identité du paiement :



* Les fonctions vérifient le nom et l’émail du client :



* + 1. **Déploiement du projet**

J’ai installé l’outil Composer sur le compte serveur et la dépendance stripe/stripe-php du projet.

Le code source a été transféré au compte serveur à l’aide du logiciel MobaXterm.

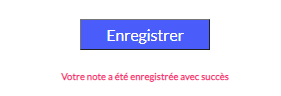
Le site web en ligne est : <https://stage.abdigitalbusiness.fr/> (site protégé par un mot de passe).

# VI

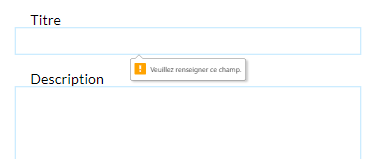
# Jeu d'essai de la fonctionnalité la plus représentative

* 1. **Projet pédagogique Blocnote**

Lorsque l’utilisateur clique sur le bouton « Ajouter une note » sur la page d’accueil, la page de l’ajout de note s’affiche. C’est un simple formulaire dont le titre et la description de la note sont à remplir. Quand l’utilisation clique sur le bouton « Enregistrer » pour valider sa note, la note sera stockée dans la base de données et le message de la réussite s’affichera comme suit :

****

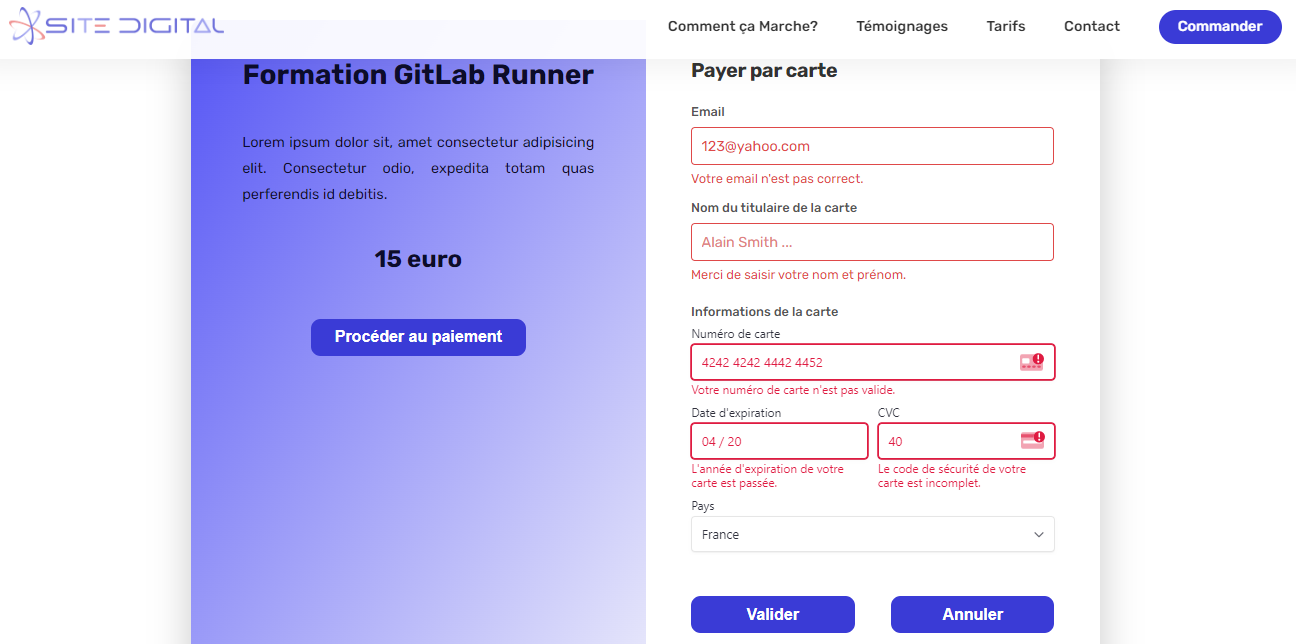
Si l’utilisateur ne remplit pas un champ de formulaire, en cliquant sur « Enregistrer », un message de demande de remplir le champ s’affiche comme suit :

****

Pour revenir à la page d’accueil, l’utilisateur peut cliquer sur le logo ou sur le nom « Bloc-Note » du site web. Et cette nouvelle note sera affichée en premier de la liste des notes.

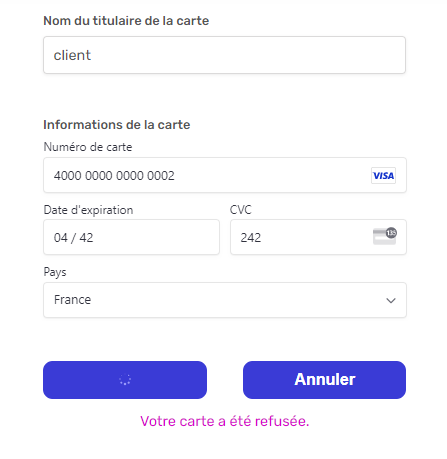
* 1. **Projet de paiement en ligne avec API Stripe**

Lors du paiement, le client doit remplir les champs « Nom » et « Email » de formulaire. Si ces champs sont vides ou mal saisis, le client sera immédiatement informé comme suivant et le bouton « Valider » reste désactivé :



L’utilisateur a également les messages d’erreur pour les champs de l’information de la carte bancaire comme ci-dessus.

Si toutes les informations sont correctes mais le paiement est refusé par la banque du client pour une raison quelconque, il aura le message :



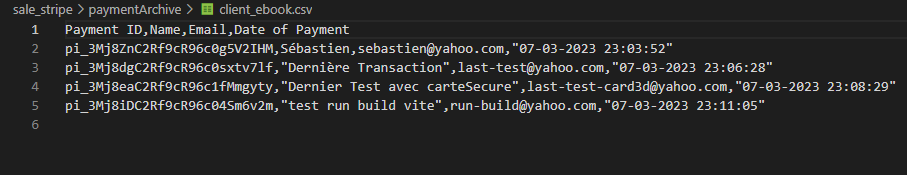
Pour la carte bancaire ayant la vérification 3D Secure, une modal de la vérification apparaît. Si la vérification est échouée, un message d’erreur s’affiche :



Finalement, lors d’un paiement réussi, le client va avoir un message de remerciement et la page sera actualisée après 3 secondes.



Et l’information du paiement sera stockée dans le fichier client\_ebook.csv :



# VII

# Veille sur les vulnérabilités de sécurité

* 1. **Projet pédagogique Blocnote**

Comme l’avoir indiqué au-dessus, le projet pédagogique est une démonstration de différentes techniques de codage destinée aux débutants. La sécurité de l’application a été contribuée via les bonnes pratiques de codage, mais je n’ai pas censé de créer les fonctions de validation.

Pourtant, l’utilisation de la fonction htmlspecialchars() de PHP a permis de nettoyer les données saisies des utilisateurs, et d’éviter donc les attaques de type « faille XSS » et « injection SQL ». De plus, l’utilisation de l’expression « placeholder » avec les fonctions bindParam() ou bindValue() dans les requêtes SQL a renforcé la sécurité contre l’attaque de type « injection SQL ».

En PHP, j’ai utilisé la méthode POST plutôt que la méthode GET pour récupérer les données des champs de formulaire. La méthode POST masque les données envoyées par l'utilisateur dans l'en-tête de la requête HTTP, ce qui les rend plus difficiles à intercepter.

La librairie ReactJS possède ses propres mesures de sécurité. Elle utilise des listes d’autorisation pour filtrer toutes les entrées d’application et inspecter le code/les fonctionnalités de l’application par rapport à la possibilité de code d’application. Les fonctionnalités insèrent des parties de code malveillantes telles que des URL ou des éléments HTML. De plus, ReactJS offre de nombreux avantages pour augmenter la vitesse du processus de développement et a amélioré les fonctionnalités de protection des applications avec des scanners de vulnérabilité et un module sérialiser-JavaScript NPM. D’ailleurs, j’ai créé les fonctions EscapeHtml() et DecodeHtml() pour nettoyer les données venant des utilisateurs, ce qui a permis de protéger les requêtes SQL de la partie back-end et d’éviter l’attaque de type « faille XSS ».

* 1. **Projet de paiement en ligne avec API Stripe**

Stripe a été certifié « [Fournisseur de services PCI de niveau 1](http://www.visa.com/splisting/searchGrsp.do?companyNameCriteria=stripe). Il s’agit du niveau de certification le plus rigoureux du secteur des paiements. Afin d’assurer ce niveau de sécurité élevé, Stripe adopte les meilleurs outils de sécurité, ainsi que les pratiques les plus efficaces en la matière :

* HTTPS et HSTS pour des connexions sécurisées : Stripe impose le protocole HTTPS pour tous les services utilisant le protocole TLS (SSL), y compris son site web public et le Dashboard pour assurer des connexions sécurisées.
* Chiffrement des données et des communications sensibles : Tous les numéros de carte font l’objet d’un chiffrement AES-256 des données au repos. Les clés de chiffrement sont sauvegardées sur des machines distinctes. Aucun des serveurs ni aucun des démons internes de Stripe ne peuvent obtenir les numéros de carte en clair.
* Divulgation de vulnérabilités via [HackerOne](https://hackerone.com/stripe) et programme de primes.

D’ailleurs, j’ai sécurisé les requêtes HTTP en utilisant les fonctions de validation et l’expression régulière RegEx. En PHP, le bloc try – catch m’a permis de capturer les erreurs de code.

# VIII

# Description d'une situation ayant nécessité une recherche

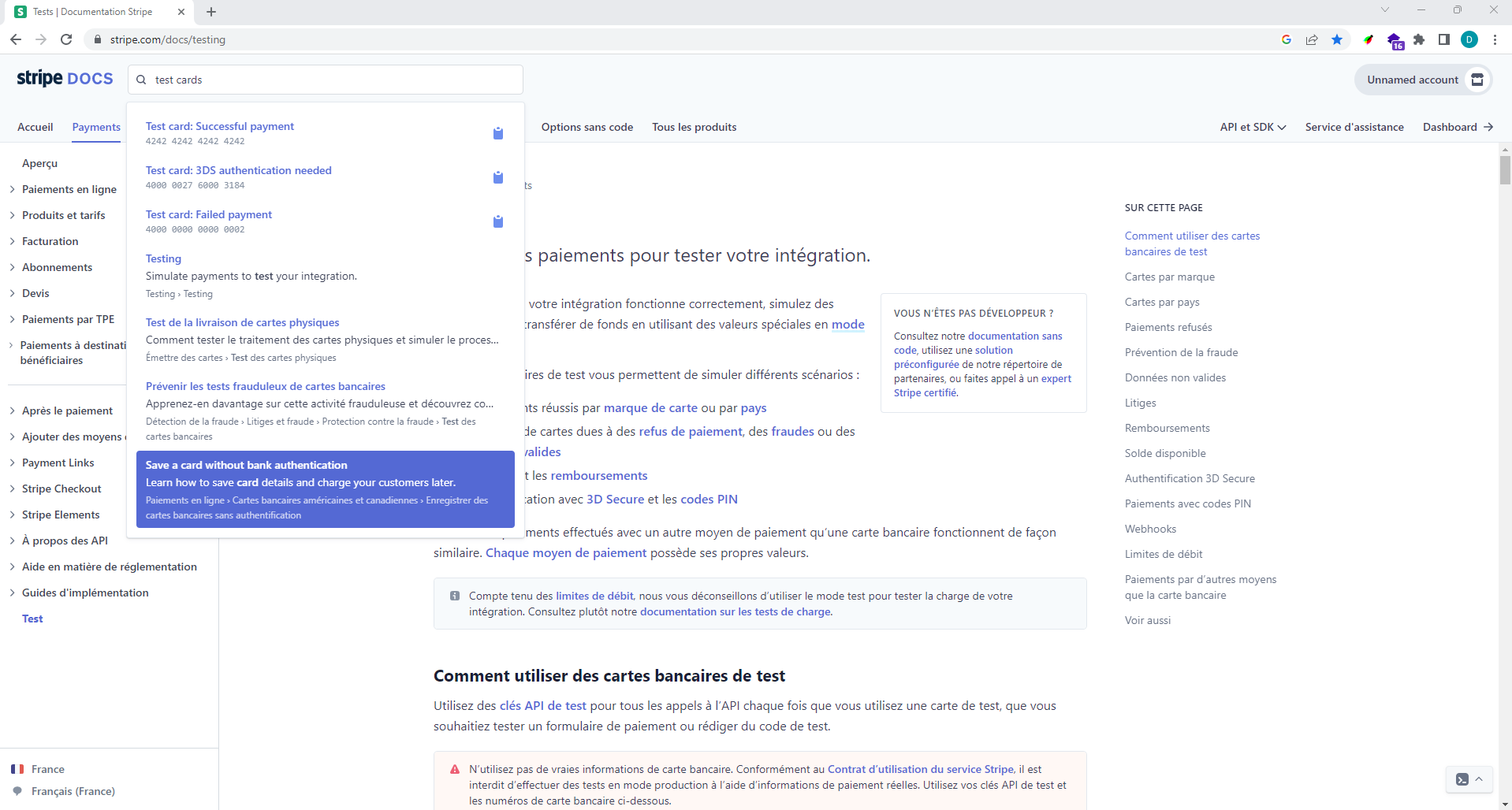
**Problématique**

Lors du projet de paiement en ligne avec API Stripe, j’ai dû tester mon application avec différentes cartes bancaires de test fournies par Stripe. Il m’a fallu les cartes bancaires de test correspondant à différentes situations : un paiement simple, un paiement avec la vérification 3D Secure, un paiement refusé à cause de la carte bancaire, etc.

**Recherche dans API Stripe**

J’ai cherché directement dans la page de l’API Stripe. Dans son document, il y a plusieurs cartes bancaires de test que l’on peut essayer (Visa, Mastercard, cartes bancaires de différents pays, etc).

Pour un test rapide, j’ai cherché « test cards » et pris les 3 cartes bancaires comme suit :

****

# IX

# Extrait du site anglophone et traduction

**Extrait du site anglophone**

The Stripe API is organized around REST. Our API has predictable resource-oriented URLs, accepts form-encoded request bodies, returns JSON-encoded responses, and uses standard HTTP response codes, authentication, and verbs.

You can use the Stripe API in test mode, which doesn't affect your live data or interact with the banking networks. The API key you use to authenticate the request determines whether the request is live mode or test mode.

The Stripe API doesn't support bulk updates. You can work on only one object per request.

The Stripe API differs for every account as we release new versions and tailor functionality. These docs are customized to your version of the API and display your test key and test data, which only you can see.

The Stripe API uses API keys to authenticate requests. You can view and manage your API keys in the Stripe Dashboard.

Test mode secret keys have the prefix sk\_test\_ and live mode secret keys have the prefix sk\_live\_. Alternatively, you can use restricted API keys for granular permissions.

Your API keys carry many privileges, so be sure to keep them secure! Do not share your secret API keys in publicly accessible areas such as GitHub, client-side code, and so forth.

Use your API key by setting it in the initial configuration of new \Stripe\StripeClient(). The PHP library will then automatically send this key in each request.

You can also set a per-request key with an option. This is often useful for Connect applications that use multiple API keys during the lifetime of a process. Methods on the returned object reuse the same API key.

All API requests must be made over HTTPS. Calls made over plain HTTP will fail. API requests without authentication will also fail.

**Traduction**

L'API Stripe est organisée autour de REST. Notre API a des URLs orientés-ressources prévisibles, accepte les corps de requête encodés en formulaire, renvoie des réponses encodées en JSON, et utilise des codes de réponse HTTP standards, une authentification et des verbes.

Vous pouvez utiliser l'API Stripe en mode test, ce qui n'affecte pas vos données en mode réel ni interagit avec les réseaux bancaires. La clé API que vous utilisez pour authentifier la requête détermine si la requête est en mode réel ou en mode test.

L'API Stripe ne prend pas en charge les mises à jour groupées. Vous ne pouvez travailler que sur un seul objet par requête.

L'API Stripe diffère pour chaque compte car nous publions de nouvelles versions et adaptons les fonctionnalités. Ces documents sont personnalisés en fonction de votre version de l'API et affichent votre clé de test et vos données de test, que vous seul pouvez voir.

L'API Stripe utilise des clés API pour authentifier les requêtes. Vous pouvez afficher et gérer vos clés API dans le Dashboard de Stripe.

Les clés secrètes de mode test ont le préfixe sk\_test\_ et les clés secrètes de mode réel ont le préfixe sk\_live\_. Vous pouvez également utiliser des clés API restreintes pour des autorisations granulaires.

Vos clés API comportent de nombreux privilèges, alors assurez-vous de les garder en sécurité ! Ne partagez pas vos clés API secrètes dans des zones accessibles au public telles que GitHub, le code du côté client, etc.

Utilisez votre clé API dans la configuration initiale de new \Stripe\StripeClient(). La librairie PHP enverra alors automatiquement cette clé dans chaque requête.

Vous pouvez également définir une clé par requête avec une option. Ceci est souvent utile pour les applications Connect qui utilisent plusieurs clés API pendant le processus. Les méthodes de l'objet renvoyé réutilisent la même clé d'API.

Toutes les requêtes API doivent être effectuées via HTTPS. Les requêtes passées via HTTP échoueront. Les requêtes API sans authentification échoueront également.