

Dossier Bilan

Troisième phase



Equipe 8

ARAVENA BRAVO Damien

JACOB Timothé

BENDRIS Nael

TIERRIE Axel

BENKHEIRA Lilya

DESCHAUX-BEAUME Maëlys

SEKMA Sarah

Synthèse du projet

Nous sommes une équipe de développeurs, nous avons été sollicités par un client dont la demande vise à représenter le patrimoine culturel en France. Nous avons donc suggéré de nous concentrer spécifiquement sur les sites patrimoniaux remarquables en Isère.

Ces sites sont des joyaux culturels et historiques dont la conservation et la valorisation représentent sur le plan historique, architectural et culturel un intérêt public.

Afin de répondre aux besoins de notre client, nous avons décidé de matérialiser cette initiative sous la forme d'une application WEB qui s'articule autour de plusieurs **fonctionnalités clés** représentées par nos **objectifs** :

Informations détaillées sur les sites patrimoniaux :

Les utilisateurs pourront bénéficier d'informations précises et détaillées concernant les sites patrimoniaux remarquables en Isère sur leurs descriptions, leurs localisation ou bien même leur attractivité étant donné que nous offrons la possibilité à nos utilisateurs de bénéficier d'un espace d'échanges.

Recommandations personnalisées :

Les utilisateurs auront la possibilité de répondre à un questionnaire afin d'obtenir des recommandations personnalisées adaptées à leurs centres d'intérêt. Ce questionnaire prendra également en compte leur situation personnelle notamment si ils sont en situation de mobilité réduite afin de proposer des lieux accessibles et inclusifs.

Réservation de guides :

Nos utilisateurs auront la possibilité de personnaliser leurs visites en choisissant parmi les guides locaux disponibles. C'est une fonctionnalité que nous avons décidé de mettre en place pour promouvoir la valorisation des sites patrimoniaux remarquables.

Bilans des risques

Gestion du travail en présentiel et à distance

Nous avons su conserver une organisation saine en combinant les séances en présentiel avec les quelques sessions en distanciel pour certains membres du groupe, cette flexibilité a été nécessaire en raison des conditions saisonnières qui ont occasionné quelques absences au sein de l'équipe projet.

Défis liés aux technologies utilisées

Une partie des technologies utilisées pour ce projet tels que React et Laravel était très peu maîtrisée par la quasi-totalité des membres du projet. Cette situation représentait donc un risque majeur pour cette dernière phase de codage. Nous avons donc mis en place des sessions de formations internes en amont de la phase de développement et également bénéficié d'une documentation de mise en place du projet élaboré par Maëlys, riche et détaillée qui nous a servi de référence tout au long du projet..

Assurance qualité

Au sein du projet, nous avons permis de mettre en place des pratiques rigoureuses concernant le développement de notre code. Ces pratiques incluent :

Utilisation d'une branche de production finale (main) : Cette branche n'est pas directement accessible par tous les membres du projet. Les modifications y sont intégrées après une vérification poussée.

Mise en place de branches spécifiques : Nous prenons également soin de développer chaque fonctionnalité clé dans une branche dédiée, ce qui nous permet de travailler indépendamment et de limiter les risques de conflits de code.

Fusion fréquentes et contrôlées : Les branches sont régulièrement fusionnées après validation, afin de maintenir un état cohérent.

Ces pratiques nous permettent d'avoir une meilleure organisation, en facilitant la collaboration entre les membres de l'équipe.

Mesures organisationnelles et techniques

Mesures organisationnelles

Pour organiser efficacement notre travail, nous avons utilisé l'outil de gestion Notion qui s'est révélée être un support indispensable pour la planification et le suivi de nos tâches. Cet outil nous a permis de :

Attribuer des responsabilités à chaque membre du groupe : Afin de garantir une clarté dans notre organisation des tâches nous avons utilisé l'outil Kanban pour leur gestion.

Centraliser les toutes les informations nécessaires : Nous avons défini explicitement nos échéances, nos objectifs et les ressources partagées sur cet outil.

Mesures techniques

Pour garantir la qualité et la robustesse de notre architecture serveur, nous avons d'abord utilisé **Postman** pour documenter et tester manuellement nos différents endpoints. Cette première phase a permis de :

Documenter chaque endpoint de manière détaillée, en spécifiant les paramètres requis, les réponses attendues et les codes de statut HTTP associés.

Tester manuellement les endpoints afin de vérifier leur fonctionnement initial, leur cohérence, et leur conformité avec les besoins métier.

Faciliter les échanges avec l'équipe front-end en fournissant une documentation claire et partagée des API.

Synthèse technique

Notre modèle de données a connu plusieurs évolutions tout au long du projet, dans le but de résoudre des problématiques découvertes en cours de développement. Ces ajustements ont impliqué des mises à jour fréquentes de la base de données nécessitant des modifications sur le backend et sur le frontend.

Voici un aperçu des problèmes posées avec nos anciennes versions :

Attributs : Nous avons constaté plusieurs incohérences dans notre base de données, notamment en raison d'un nombre important d'attributs redondants dans les tables Guide et Utilisateur. Pour résoudre ce problème et gérer au mieux les différences de rôles entre les guides et les utilisateurs, nous avons décidé de factoriser ces attributs dans une table commune. Il y a ainsi un lien entre un guide et un utilisateur grâce à l'identifiant unique présent dans la table Utilisateur.

Pour garantir une mise en oeuvre technique cohérente, nous avons mis en place les éléments suivants :

Laravel en tant qu'API : Nous avons pris la décision d'utiliser le framework Laravel pour PHP, il nous a permis de gérer les échanges avec la base de données PostgreSQL et de sérialiser les modèles en JSON pour simplifier l'intégration avec le côté client.

React côté client : Nous avons choisi React comme technologie principale pour l'interface utilisateur, elle dispose d'une intégration du bundle Vite qui inclut plusieurs packages essentiels comme React et TypeScript. Après l'installation des dépendances via npm, le projet est lancé localement.

Nous utilisons **MaterialUI**, qui grâce à sa bibliothèque nous permet d'intégrer des composants prêts à l'emploi et conformes aux bonnes pratiques de design. Cela nous a permis d'accélérer le développement de l'interface utilisateur.

Bilan général

Pour ce bilan nous allons énumérer nos réussites, nos défis rencontrés et les perspectives futures.

Réussites : Notre application permet de répondre aux besoins de l'utilisateur à savoir pouvoir représenter le patrimoine culturel en France, grâce à des fonctionnalités telles que les recommandations personnalisées et la réservation de guide.

Grâce à l'intégration de React et Material-UI, nous offrons un design moderne, facilitant l'exploration des sites patrimoniaux remarquables.

L'utilisation de l'outil Notion a permis dès la première phase de structurer le travail d'équipe efficacement, attribuer des responsabilités.

Défis rencontrés : Nous avons dû nous adapter à des outils peu maîtrisés en début de projet, ce qui a nécessité des formations internes au sein de l'équipe. Le peu de temps qui nous a été accordé pour le développement ne nous a pas permis de développer toutes les fonctionnalités voulues et donc contraint à mitiger certaines de nos fonctionnalités..

Perspectives futures : Nous envisageons d'implémenter la fonctionnalité qui permettrait aux utilisateurs de se proposer en tant que guide avec un système de validation.

Nous suggérons également d'ajouter des fonctionnalités multilingues pour rendre l'application accessible à un plus grand public.

Annexes

Structure du code

La structure de notre application Dauphinéo a été pensée pour garantir une organisation claire. Cette section décrit un **aperçu** de l'organisation des fichiers et dossiers. Nous visons à travers cette partie permettre une compréhension rapide du projet.

Backend : Notre dossier backend contient le code de l'API, développée avec Laravel qui gère les relations entre les entités et l'accès aux données. voici un aperçu des sous dossiers principaux.

```
Backend/
├── app/
│   ├── Http/
│   │   └── Controllers/
│   └── Models/
├── config/
└── routes/Api.php
```

App/ : Contient les composants principaux de l'application.

Http/Controllers : Regroupe tous nos controllers qui communiquent avec la base de données.

Models/ : Définit les modèles comme SPR, Guide et Disponibilités.

Config/ : Contient tous les fichiers de configurations essentiels, il les centralise toutes avec notamment des configurations pour le système d'authentification.

routes/Api.php : Nous définissons dans ce fichier les routes API de l'application.

Frontend : Notre dossier frontend contient toute l'architecture React de l'application, organisée pour faciliter sa maintenabilité.

```
frontend/
├── src/
│   ├── pages/
│   ├── utils/
│   │   └── axiosInstance.tsx
│   ├── components/
│   ├── api/
│   └── endpoints/
```

pages/ : Contient toutes les pages relatives à notre application Dauphinéo

utils/ : Contient un fichier axiosInstance.tsx qui définit les paramètres du headers à envoyer à chaque requête

components/ : Contient tous les composants relatifs à notre application Dauphinéo

api/ : Contient un répertoire **/endpoints** qui définit les requêtes où vont s'acheminer les données (routes de l'API Laravel)

README

Rôle et importance : Le README est la carte de visite de Dauphinéo. En effet, il permet de :

- Donner le premier aperçu du projet aux nouveaux arrivants
- Guider les développeurs dans l'installation et la configuration
- Présenter l'architecture technique de manière synthétique

Structure du document : Le README débute par une présentation concise de Dauphinéo, accompagnée des badges technologiques qui illustrent instantanément le stack technique utilisé.

On trouve ensuite, une table des matières structurant le document en sections claire :

- À propos
- Stack technique
- Installation
- Architecture du projet
- Configuration
- Documentation API

Plus simplement, le README contient toutes les informations nécessaires à la compréhension, à l'installation et à l'utilisation de l'application Dauphinéo.

Voici un lien qui redirige vers le README : [Repository rendus - GitLab](#)

Cahier des tests

Dans le cadre de l'évaluation des interfaces de notre application Dauphinéo, nous avons effectué des tests utilisateurs en utilisant le **System Usability Scale** (SUS).

Cet outil standardisé nous a permis de mesurer l'utilisabilité perçue par les utilisateurs. Ce questionnaire évalue donc plusieurs aspects clés tels que la facilité d'utilisation, la cohérence de nos fonctionnalités et la satisfaction en sa globalité.

A travers ce questionnaire, les résultats obtenus nous ont permis d'identifier clairement les points forts ainsi que les éléments et logique des fonctionnalités à optimiser de nos interfaces.

Utilisateur n° 1 : Nous avons sollicité un proche pour un test utilisateur

Question	Réponse
Je pense que j'aimerais utiliser ce système fréquemment.	4
J'ai trouvé le système inutilement complexe.	1
J'ai trouvé ce système facile à utiliser.	5
Je pense que j'aurais besoin d'un support technique pour être capable d'utiliser un système.	1
J'ai trouvé que les différentes fonctions de ce système étaient bien intégrées.	3
J'ai trouvé qu'il y avait trop d'incohérence dans ce système.	1
Je suppose que la plupart des gens apprendraient très rapidement à utiliser ce système.	3
J'ai trouvé ce système très contraignant à utiliser.	1
Je me suis senti(e) très confiant(e) en utilisant ce système.	4
J'ai dû apprendre beaucoup de choses avant de me familiariser avec le système.	1

Le score SUS a été calculé comme suit : Nous avons attribué pour les questions impaires la note de : **score - 1** et nous avons attribué pour les questions paires la note de : **5 - score**

Donc d'après cette référence, le calcul détaillé de l'utilisateur n°1 se présente de cette manière :

Question n° 1 : $4-1 = 3$

Question n° 2 : $5-1 = 4$

Question n° 3 : $5-1 = 4$

Question n° 4 : $5-1 = 4$

Question n° 5 : $3-1 = 2$

Question n° 6 : $5-1 = 4$

Question n° 7 : $3-1 = 3$

Question n° 8 : $5-1 = 4$

Question n° 9 : $4-1 = 3$

Question n°10 : $5-1 = 4$

Ces résultats nous permettent d'aboutir au score final qui se calcule comme suit :

Score final représente la somme des résultats multipliés par 2.5 pour obtenir un score sur 100.

Nous établissons donc le score à 34 multiplié par 2.5, ce qui nous permet d'obtenir un score SUS de **85**. Ce résultat permet à Dauphineo de se positionner clairement parmi les systèmes les mieux notés en termes d'utilisabilité.

Utilisateur n ° 2 : Nous avons sollicité un proche pour un test utilisateur

Question	Réponse
Je pense que j'aimerais utiliser ce système fréquemment.	3
J'ai trouvé le système inutilement complexe.	2
J'ai trouvé ce système facile à utiliser.	4
Je pense que j'aurais besoin d'un support technique pour être capable d'utiliser un système.	2
J'ai trouvé que les différentes fonctions de ce système étaient bien intégrées.	2
J'ai trouvé qu'il y avait trop d'incohérence dans ce système.	3
Je suppose que la plupart des gens apprendraient très rapidement à utiliser ce système.	4
J'ai trouvé ce système très contraignant à utiliser.	2
Je me suis senti(e) très confiant(e) en utilisant ce système.	3
J'ai dû apprendre beaucoup de choses avant de me familiariser avec le système.	2

Ces résultats nous permettent d'aboutir au score final qui se calcule comme suit :

Nous établissons donc le score à 27 multiplié par 2.5 ce qui nous permet d'obtenir un score SUS de **27**. L'utilisateur trouve le système globalement facilement accessible, mais il a relevé des défauts mineurs. Bien que le système soit jugé facile d'utilisation, certaines fonctionnalités manquent de cohérence. Ce retour souligne un bon potentiel d'améliorations.

Perspectives d'améliorations :

A partir des remarques collectées lors des tests utilisateurs, nous pouvons émettre plusieurs axes d'améliorations. Tout d'abord nous optons pour la simplifications des dates de saisie car le champ de saisie des dates est jugé peu pratique par l'utilisateur n°1. Nous envisageons de le remplacer par un calendrier interactif permettant donc une sélection plus intuitive des dates.

Nous observons également que le champ de réservation de guide n'est pas suffisamment visible et explicite, ce qui peut perdre l'utilisateur. Nous envisageons de revoir le design pour rendre ce champ plus discernable.

Nous prévoyons d'intégrer des tests utilisateurs itératifs pour valider les améliorations futures avant leur déploiement et adapter nos interfaces pour permettre une implémentation progressive sur différents formats d'écran pour garantir une cohérence et une utilisabilité optimale pour tous les appareils.

Traitement de données

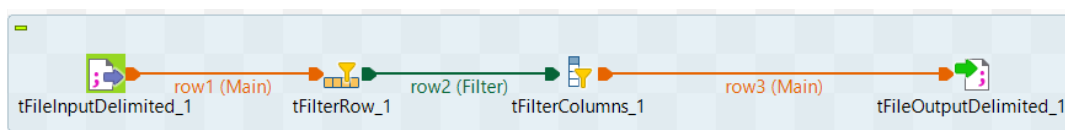
Pour cette partie, nous avons récupéré les données des SPR au format CSV sur le site **data.gouv.fr**.

Notre objectif étant de représenter le patrimoine culturel en France nous avons dans un premier temps filtrer les données pour ne conserver que les SPR situés en Isère. Ensuite nous avons supprimé les colonnes inutiles et ajouté celles qui étaient nécessaires.

Nous avons également complété les données en ajoutant des descriptions pour chaque SPR, des mots-clés descriptifs utiles pour la fonctionnalité du questionnaire, une estimation du temps de visite, ainsi que la capacité maximale d'accueil pour une visite.

Vous trouverez ci-joint un aperçu de la pipeline Talend qui se compose des étapes suivantes :

- Extraction des données : Lecture du CSV et validations des formats d'entrées
- Transformation : Conserver les SPR d'isères et suppression des colonnes non pertinentes. Intégration d'attributs (descriptions , mots-clés, capacité)
- Chargement : Export au format approprié (CSV) pour l'intégration dans notre application Dauphinéo.



Pipeline Talend

Données traitées :

Une fois le traitement effectué, seulement 20 SPR ont été conservés. Par ailleurs l'enrichissement de nos données notamment avec l'ajout de certains attributs tels que les mots-clés a permis de mettre en place l'API responsable du questionnaire.

Check-list de sécurité

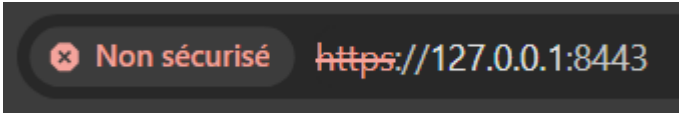
Vulnérabilité dans les logiciels utilisés :

Les mises à jour de sécurité pour les logiciels fournis par Debian sont faites par un `sudo apt upgrade` manuellement. Pour les automatiser nous avons installé le package `unattended-upgrades` pour garder la VM actualisée avec les dernières mises à jour de sécurité.

Confidentialité et intégrité des données circulant sur les réseaux :

Nous avons mis en place un certificat auto-signé HTTPS avec OpenSSL

```
root@debian:~# cat /etc/apache2/sites-available/default-ssl.conf
<VirtualHost *:443>
```



Notre base de données utilise bien le protocole TLS et nous avons créé un rôle Laravel disposant des permissions nécessaires.

```
root@debian:~# psql -U laravel -d dauphineo -h 192.168.14.215
Password for user laravel:
psql (15.10 (Debian 15.10-0+deb12u1))
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, compression: off)
Type "help" for help.

dauphineo=> SELECT * FROM pg_stat_ssl WHERE pid = pg_backend_pid();
 pid | ssl | version |      cipher      | bits | client_dn | client_serial | issuer_dn
-----+----+-----+-----+-----+-----+-----+-----
179549 | t   | TLSv1.3 | TLS_AES_256_GCM_SHA384 | 256 |           |              |
(1 row)
```

Base de données Dauphinéo

Contrôle d'accès :

Dans le fichier `pg_hba.conf`, nous avons configuré les accès à la base de données pour que seul le rôle "laravel", depuis la VM web, puisse se connecter à Dauphineo via SSL.

Nous avons ensuite installé puis configuré le pare feu UFW :

```
root@debian:~# ufw status numbered
Status: active
```

To	Action	From
---	-----	----
[1] 443/tcp	ALLOW IN	Anywhere
[2] 5432	ALLOW IN	192.168.14.115
[3] 80/tcp	ALLOW IN	Anywhere
[4] 22/tcp	ALLOW IN	Anywhere
[5] 443	ALLOW IN	Anywhere
[6] 443/tcp (v6)	ALLOW IN	Anywhere (v6)
[7] 80/tcp (v6)	ALLOW IN	Anywhere (v6)
[8] 22/tcp (v6)	ALLOW IN	Anywhere (v6)
[9] 443 (v6)	ALLOW IN	Anywhere (v6)

```
root@debian:~# ufw status numbered
Status: active
```

To	Action	From
---	-----	----
[1] 5432/tcp	ALLOW IN	192.168.14.115
[2] 22/tcp	ALLOW IN	Anywhere
[3] 22/tcp (v6)	ALLOW IN	Anywhere (v6)

Enfin, nous avons gardé les configurations par défaut *Fail2ban* pour sécuriser les connexions SSH :

```
root@debian:~# fail2ban-client status
Status
|- Number of jail:      1
`- Jail list:  sshd
root@debian:~# fail2ban-client get sshd maxretry
5
root@debian:~# fail2ban-client get sshd findtime
600
root@debian:~# fail2ban-client get sshd bantime
600
```

Statut du client fail2ban

Stockage des mots de passe avec hachage (obligatoire), salage (indispensable) et poivrage (bon à avoir) :

Les mots de passe envoyés par les utilisateurs sont hashés automatiquement grâce au cast *hashed* de Laravel. Celui-ci utilise la méthode *Hash::make()*. Par défaut, l'algorithme de hachage est **Bcrypt**, mais nous l'avons modifié pour avoir **Argon2id**.

Nous allons faire un point sur les vulnérabilités dans les logiciels développés :

Utilisation exclusive de requêtes préparées :

Eloquent repose sur le composant **Query Builder** de Laravel, qui, à son tour, utilise PDO (PHP Data Objects) pour exécuter toutes les interactions avec la base de données. PDO s'appuie sur des requêtes préparées pour toutes les requêtes, qu'elles soient générées automatiquement ou manuellement.

Filtrage des données saisies par les utilisateurs :

La validation des données saisies par les utilisateurs repose principalement sur la méthode *validate()* (ou les classes personnalisées de validation). Ce système garantit que les données reçues sont filtrées, sécurisées et conformes aux règles métier définies avant de les utiliser ou de les stocker.

Jeton de validité dans les formulaires :

Etant donné que notre backend fait office d'API, elle est consommée par le client. Notre stratégie est donc celle des **cookies HTTP-only** pour gérer la session et associer chaque demande d'action avec un **token CSRF**. Ce cookie est utilisé pour transmettre le jeton CSRF via l'en-tête **X-XSRF-TOKEN** dans les requêtes de l'API. Avec cette méthode gérée par Laravel Sanctum, notre application est protégée contre les attaques **CSRF**.

Configuration de déploiement

Nous allons vous montrer le fichier qui lance les différentes étapes de déploiement de l'application :

```
# Étape 5: Compiler le frontend
cd $FRONTEND_DIR
echo "Compilation des assets React..."
npm run build

echo "Configuration des permissions..."
sudo chown -R www-data:www-data $BACKEND_DIR/storage
sudo chmod -R 775 $BACKEND_DIR/storage

# Étape 6: Redémarrer Apache
echo "Redémarrage du serveur Apache..."
sudo systemctl restart apache2

# Étape 7: Redémarrer le serveur Laravel
echo "Démarrage du serveur Laravel..."
cd $BACKEND_DIR
#php artisan serve &

echo "Déploiement terminé!"

#!/bin/bash

# Définir les répertoires
PROJECT_DIR="/var/www/html/rendus"
BACKEND_DIR="$PROJECT_DIR/backend"
FRONTEND_DIR="$PROJECT_DIR/frontend"

# Arrêter le serveur Laravel existant
echo "Arrêt du serveur Laravel..."
#pkill -f "php artisan serve" || true

# Aller dans le répertoire du projet
cd $PROJECT_DIR

# Étape 1: Mettre à jour le dépôt
echo "Mise à jour du dépôt..."
git pull origin main

# Étape 2: Installer les dépendances PHP
echo "Installation des dépendances PHP..."
cd $BACKEND_DIR
composer install

# Nettoyage du cache Laravel
php artisan config:clear
php artisan cache:clear
php artisan route:clear
php artisan view:clear

# Étape 3: Installer les dépendances NPM
echo "Installation des dépendances Node.js..."
cd $FRONTEND_DIR
npm install

# Étape 4: Mettre à jour les migrations Laravel
echo "Exécution des migrations Laravel..."
cd $BACKEND_DIR
php artisan migrate --force
php artisan serve --port=8000 >/dev/null 2>&1 &

#php artisan serve &

# Étape 5: Compiler le frontend
cd $FRONTEND_DIR
```

Fichier de déploiement

Configuration du serveur Apache :

Tout d'abord, les types MIME ont été configurés pour inclure les fichiers JavaScript, CSS, SVG et JSON, ce qui garantit leur correcte interprétation par les navigateurs.

Ensuite, le Cross-Origin Resource Sharing (CORS) a été activé afin de permettre les requêtes provenant de l'origine `https://127.0.0.1:8443`. Les méthodes HTTP autorisées incluent GET, POST, PUT, DELETE et OPTIONS. Des en-têtes spécifiques ont également été ajoutés pour gérer les autorisations et l'utilisation de tokens.

Une gestion des redirections a été mise en place pour React Router, avec une réécriture des URLs redirigeant toutes les requêtes vers le fichier `index.html`. Pour le back-end, une configuration proxy a été définie pour les requêtes vers `/api`, redirigeant celles-ci vers un serveur Laravel à l'adresse `http://127.0.0.1:8000/api`. Des en-têtes supplémentaires ont été ajoutés pour gérer le cache et les réponses JSON, tout en forçant les navigateurs à ne pas stocker les réponses en cache.

Enfin, la gestion SSL a été configurée à l'aide d'un certificat autosigné (**selfsigned.crt**) et d'une clé privée (**dauphineo.key**), assurant une connexion sécurisée via HTTPS.

```

# Add proper MIME types for modern web applications
<IfModule mod_mime.c>
    AddType application/javascript .js .mjs
    AddType text/javascript .js .mjs
    AddType text/css .css
    AddType image/svg+xml .svg
    AddType application/json .json
</IfModule>

# Enable CORS for development
<IfModule mod_headers.c>
    Header set Access-Control-Allow-Origin "https://127.0.0.1:8443"
    Header set Access-Control-Allow-Methods "GET, POST, PUT, DELETE, OPTIONS"
    Header set Access-Control-Allow-Headers "X-Requested-With, Content-Type, X-Token-Auth, Authorization"
    Header set Access-Control-Allow-Credentials "true"
</IfModule>
</VirtualHost>

```

```

<VirtualHost *:443>
    ServerAdmin webmaster@dauphineo.fr
    ServerName www.dauphineo.fr
    DocumentRoot /var/www/html/rendus/frontend/dist/

    # Available loglevels: trace8, ..., trace1, debug, info, notice, warn,
    # error, crit, alert, emerg.
    # It is also possible to configure the loglevel for particular
    # modules, e.g.
    #LogLevel info ssl:warn

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
    ProxyRequests Off
    ProxyPreserveHost On

    ProxyPass /api http://127.0.0.1:8080/api
    ProxyPassReverse /api http://127.0.0.1:8080/api
    # Configuration pour l'API Laravel
    <Location /api>
        LogLevel debug proxy:trace5
        DirectoryIndex disabled

        Require all granted
        Header always set Content-Type "application/json"
        Header always set Cache-Control "no-cache, no-store, must-revalidate"
        Header always set Pragma "no-cache"
        Header always set Expires "0"

    # CORS Headers
        Header always set Access-Control-Allow-Origin "https://127.0.0.1:8443"
        Header always set Access-Control-Allow-Methods "GET, POST, PUT, DELETE, OPTIONS"
        Header always set Access-Control-Allow-Headers "X-Requested-With, Content-Type, X-Token-Auth, Authorization, X-XSRF-TOKEN"
        Header always set Access-Control-Allow-Credentials "true"

        RewriteEngine On
        RewriteCond %{REQUEST_METHOD} OPTIONS
        RewriteRule "(.*)$ $1 [R=200,L]"
    </Location>

    # Redirection pour React Router
    #Alias / /var/www/html/rendus/frontend/dist/
    <Directory /var/www/html/rendus/frontend/dist>

```

```

        </Location>
        # Redirection pour React Router
        #Alias / /var/www/html/rendus/frontend/dist/
        <Directory /var/www/html/rendus/frontend/dist>
            Options -MultiViews
            AllowOverride All
            Require all granted

            RewriteEngine On
            RewriteBase /
            RewriteCond %{REQUEST_FILENAME} !-f
            RewriteCond %{REQUEST_FILENAME} !-d
            RewriteRule "(.*)$ index.html [QSA,L]"

            # Proper MIME type handling
            <FilesMatch "\.js$">
                Header set Content-Type "application/javascript"
            </FilesMatch>
            <FilesMatch "\.css$">
                Header set Content-Type "text/css"
            </FilesMatch>
        </Directory>

        # For most configuration files from conf-available/, which are
        # enabled or disabled at a global level, it is possible to
        # include a line for only one particular virtual host. For example the
        # following line enables the CGI configuration for this host only
        # after it has been globally disabled with "a2disconf".
        #Include conf-available/serve-cgi-bin.conf

        # SSL Engine Switch:
        # Enable/Disable SSL for this virtual host.
        SSLEngine on

        # A self-signed (snakeoil) certificate can be created by installing
        # the ssl-cert package. See
        # /usr/share/doc/apache2/README.Debian.gz for more info.
        # If both key and certificate are stored in the same file, only the
        # SSLCertificateFile directive is needed.
        #SSLCertificateFile /etc/ssl/certs/ssl-cert-snakeoil.pem
        #SSLCertificateKeyFile /etc/ssl/private/ssl-cert-snakeoil.key
        SSLCertificateFile /etc/apache2/ssl/selfsigned.crt
        SSLCertificateKeyFile /etc/apache2/ssl/dauphineo.key
    </Directory>

```

Aperçu du fichier **default_SSL.conf**