**DOCUMENTATION IAPEL**

**SOMMAIRE**

Table des matières

[**1.** **INTRODUCTION** 3](#_Toc201916384)

[1.1 Le projet 3](#_Toc201916385)

[**2** **MISE EN PLACE** 3](#_Toc201916386)

[2.1 PRÉREQUIS 3](#_Toc201916387)

[2.2 INSTALLATION 3](#_Toc201916388)

[ Copier le projet 3](#_Toc201916389)

[ Exécuter 3](#_Toc201916390)

[ Lancement du serveur 4](#_Toc201916391)

[2.3 CONFIGURATION 4](#_Toc201916392)

[**3** **STRUCTURE DU PROJET** 4](#_Toc201916393)

[3.1 App 4](#_Toc201916394)

[3.2 Bootstrap 4](#_Toc201916395)

[3.3 Config 4](#_Toc201916396)

[3.4 Database 4](#_Toc201916397)

[3.5 Routes 5](#_Toc201916398)

[3.6 Storage 5](#_Toc201916399)

[**4** **EXPLICATION** 6](#_Toc201916400)

[4.1 STOCKAGE 6](#_Toc201916401)

[4.2 ENVIRONNEMENT 6](#_Toc201916402)

[4.3 LAYOUT 7](#_Toc201916403)

[4.4 JSON 8](#_Toc201916404)

[*4.4.1* LES DONNÉES JSON POUR LES DEVIS 8](#_Toc201916405)

[*4.4.2* LES DONNÉES JSON POUR LES RAPPORT / CERFA 8](#_Toc201916406)

[*4.4.3* LES DONNÉES JSON POUR LES TOKENS 8](#_Toc201916407)

[**5** **SÉCURITÉ & AUTHENTIFICATION** 8](#_Toc201916408)

[5.1 TOKEN 8](#_Toc201916409)

[5.2 HEADER 9](#_Toc201916410)

[5.3 MIDDLEWARE & ACCÈS 9](#_Toc201916411)

[**6** **FONCTIONNALITÉS** 9](#_Toc201916412)

[6.1 RESSEMBLANCE MODULE DEVIS & RAPPORT D’INTERVENTION 9](#_Toc201916413)

[6.2 MODULE DEVIS 9](#_Toc201916414)

[6.3 MODULE RAPPORT D’INTERVENTION 10](#_Toc201916415)

[**7** **ROUTES** 12](#_Toc201916416)

# **INTRODUCTION**

## Le projet

IAPEL est un portail web permettant au client de APEL et APEL MOBILE de signer leur devis en ligne et de rédiger leur rapport d’intervention. Le portail permet de faciliter l’accès au devis pour les clients des entreprises via une URL et de faciliter l’accès au rapport d’intervention pour les techniciens.

Les devis sont importés depuis APEL BATIMENT, un lien pour y accéder est créé par IAPEL, le lien est envoyé par mail au client de l’entreprise pour pouvoir le signer et le certifier.

Les rapports d’intervention sont importés depuis APEL MOBILE pour pouvoir être remplis et signés. Le technicien peut à tout moment revoir son rapport d’intervention.

# **MISE EN PLACE**

## PRÉREQUIS

* Le projet, [GitHub](https://github.com/El2iInformatique/iapel)
* Composer (gestionnaire de package), [ici](https://getcomposer.org/)
* PHP 8.x : [Windows](https://windows.php.net/download#php-8.4), [autre](https://www.php.net/downloads.php)
* Python, [ici](https://www.python.org/)
* Postman pour les tests, [ici](https://www.postman.com/)

## INSTALLATION

### Copier le projet

1. *git clone* [*https://github.com/El2iInformatique/iapel.git*](https://github.com/El2iInformatique/iapel.git)

### Exécuter

1. Exécuter (long) : *composer install*
2. En parallèle, copier le fichier .env.exemple et le renommer en .env : *cp .env.exemple .env*
3. Exécuter : *php artisan key :generate*
4. Exécuter : *php artisan migrate*
5. Exécuter : *php artisan storage:link*

### Lancement du serveur

php artisan serve

## CONFIGURATION

Une image contenant texte, Police, capture d’écran, conception

Le contenu généré par l’IA peut être incorrect.Dans le fichier .env vérifier que sqlite est utiliser comme base de données.

Ajouter cette ligne dans le ficheir .env , elle servira pour les requêtes avec vérification du header.



# **STRUCTURE DU PROJET**

Les dossiers app, bootstrap, config, database, routes, storage se trouve à la racine du projet

## App

Contient les dossier http, Models, Providers

* **Http** contient lui-même les dossiers Controllers et Middleware
  + **Controllers** contient tous les Controller utiliser dans le projet (6)
  + **Middleware** contient tous les Middleware utiliser dans le projet (4)
* **Models** contient tous les models du projet
* **Providers** contient tous les providers du projet

## Bootstrap

Contient le fichier app.php qui permet d’ajouter ou supprimer des Middleware / route ; fichier de configuration de bootstrap.

## Config

Contient tous les fichiers de configuration du projet comme secrets.php qui récupère tous les mots de passe des clients depuis le fichier .env. Voir partie configuration. D’autre fichiers de configuration comme app.php ou auth.php pour les authentifications s’y trouve.

## Database

Contient la base de données sqlite « database.sqlite » utiliser partout dans le projet et les fichiers de migration de la base de données. Ces fichiers sont utilisés par laravel lors de l’exécution de la commande *php artisan migrate*.

Les tables utilisées par IAPEL :

* **token**, utilisé par les tokens pour trouver le chemin d’un document :
* **id** (INT PRIMARY KEY AUTO\_INCREMENT)
* **token** (VARCHAR), la chaîne de caractères
* **paths** (VARCHAR), chemin relatif vers le document
* **type\_document** (VARCHAR), type du document
* **expires\_at** (DATETIME), date d’expiration du token
* **created\_at** (DATETIME), date de création du token
* **used** (TINYINT(1), BOOLEAN), permet de savoir si le token a déjà été utilisé, utile que pour les devis
* **layout\_client**, utilisé par les layout pour trouver le nom des layout et vérifier que le client en utilise :
* **id** (INT PRIMARY KEY AUTO\_INCREMENT)
* **nom\_layout** (VARCHAR), nom du client
* **nom\_client** (VARCHAR), nom du layout

## Routes

Contient tous les fichiers des routes. web.php contient les routes principales du projet et les définitions des routes d’api, devis, rapport, generate. Le dossier « **principale** » contient les fichiers devis, generate, rapport-cerfa qui sont utiliser pour séparer les différentes routes.

* **Devis.php** définie un groupe de route « devis », dans se groupe sont créer plusieurs routes en rapport avec les devis comme signature ou delete-devis
* **Generate.php** définie un groupe de route « generate », dans se groupe sont créer plusieurs routes qui permettent de générer des PDF en rapport avec le nom des routes.
* **Rapport-cerfa.php** définie un groupe de route « rapport-cerfa », dans se groupe sont créer plusieurs routes en rapport avec les rapports d’intervention et les cerfas.

## Storage

Contient tous les fichiers qui doivent être stocker coté serveur. Dans ce dossier il y a app, Framework et logs.

* **App** contient private, public et signature qui sont utiliser pour :
* **Private** contient toutes les données qui ne doivent pas être accessible aux clients
* **Public** contient toutes les données qui peuvent être utilisés par le client
* **Signature** contient le certificat pour certifié le devis et le script python qui certifie le devis
* **Framework** utiliser par le Framework
* **Logs** et le dossier où se trouve les logs dans le fichier laravel.log

# **EXPLICATION**

Quand on parle de « fichier JSON », on parle du fichier de sauvegarde que tous les documents enregistrés dans IAPEL possèdent. Il a le même nom que le document / uid dans le même dossier que le document. Il permet de stocker les informations nécessaires au document comme ces informations ou les éléments nécessaires au token.

## STOCKAGE

Dans le dossier d’un document, on retrouve toujours un fichier JSON qui stocke les informations liées au document et les informations utiles au token. Le stockage de tout les dossier client se fait dans iapel/storage/app/public.

Pour les rapports d'intervention, un modèle de PDF se trouve dans {client}/rapport\_intervention/rapport\_intervention.pdf,

Chemin : iapel/sotckage/app/public/{client}/{document/{uid}, uid étant le nom du document, document étant le type de document.

Les logos des clients se trouvent dans {client}/logo.png .

Une image contenant texte, capture d’écran, Police, nombre

Le contenu généré par l’IA peut être incorrect. **ARBORESCENCE D’UN DOSSIER CLIENT (Apple) :**

## ENVIRONNEMENT

Le fichier environnement **« .env »** contient tous les mots de passe des clients et admins de IAPEL, ce mot de passe n’est utilisé que dans des cas où il faut supprimer le document ou en ajouter. Dans le tableau des routes, il sera écrit si la route a besoin d’un mot de passe. Ce mot de passe doit être inclus dans le header de la requête sous le nom de « secret-token » . Pour ajouter de nouveaux mots de passe, il suffit de mettre le nom du client avec le mot de passe et dans secret.php ajouter le mot de passe au dictionnaire. Ex : CLIENT=MOTDEPASSE.

Dans .env :



Dans secret.php



## LAYOUT

Les layouts sont utilisés dans la partie rapport d’intervention et servent à ajouter une partie 4 dans la VUE. Cette partie est personnalisable par le client mais est pour l’instant créée par nous. Tout le contenu de la partie 4 est contenu dans un fichier dans vue/custom/{client}\_LAYOUT.blade.php. La relation entre le layout et le nom du client est stockée en BDD dans la table layout\_client :

* Nom\_client, contient le nom du client
* Nom\_layout, contient le nom du layout. Ce nom doit contenir le nom de l’entreprise exacte + « \_LAYOUT » car il est relié au fichier du même nom dans vue/custom.

Le layout est inclus dans la vue bi.blade.php grâce à un @if qui vérifie si un layout existe. La vérification se fait dans le BiController qui vérifie si un layout existe avec le nom du client dans la BDD. Le layout est ajouté dynamiquement avec un @include dans la vue

Le fonctionnement des layouts niveau stockage se fait avec 3 clés :

* value : valeur ; value stock l’information donner par le technicien
* question : valeur ; question est la question a afficher dans le PDF
* type : type ; type est le type de la donnée (image, texte , select)

Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.Lors de la récupération des données dans le BiControllers, les données sont stockées dans le tableau JSON « **complement**\_**client** » dans le fichier JSON sous forme :

Lors de la génération du PDF, une boucle for affiche automatiquement avec le bon écart les texte / images.

* Les layouts peuvent contenir des inputs de type texte et d'image.
* Ils doivent être placés dans le dossier vue/custom
* Doivent être au format ***[NOM\_CLIENT]\_LAYOUT.blade.php***
* Doivent être reliés dans la base de données dans la table layout\_client à un client.

Un script Python (add\_layout.py) a été créé pour faciliter la création de layout. Il faut juste lancer le script en administrateur, donner le nom du client, le nombre d’item (input / question) et voilà le script ajoute automatiquement les données en base de données et crée le fichier **.balde.php**. Il reste seulement à modifier le fichier **.balde.php** avec le bon nom des questions et le bon type d’input.

Le script Python se trouve à la racine du projet.

## JSON

On accède au fichier JSON grâce au token qui donne le chemin depuis le stockage vers le document.

Une image contenant capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.Le fichier JSON est créé lors de l’appel de la route /api/generate-token-devis ou /api/ generate-token-rapport-cerfa et récupère toutes les données et les enregistre. Pour accéder au JSON que ce soit depuis les devis ou rapports d'intervention / cerfa, c’est le même moyen mais le contenu est très différent à part la partie token qui est la même, voir **image**.

### LES DONNÉES JSON POUR LES DEVIS

Voici une explication détaillée de toutes les données qui peuvent être trouvées dans le fichier JSON d’un devis

* **organisation\_id** : nom de l’entreprise / client
* **devis\_id** : le nom du devis
* **Tiers** : /
* **client\_email** : mail du client
* **Titre** : titre du devis
* **montant\_HT** : montant hors taxe du devis
* **montant\_TVA** : montant avec la TVA du devis
* **montant\_TTC** : montant, toute taxe confondue, du devis
* **coords** : Liste JSON
* **x\_signature** : coordonner X de la signature sur le document, coin haut de l’image
* **y\_signature** : coordonner Y de la signature sur le document, coin gauche de l’image
* **x\_date** : coordonner X de la date sur le document
* **y\_date** : coordonner Y de la date sur le document
* **nb\_pages** : Nombre de pages depuis la dernière page où va être mise la signature. Si un document a 4 pages et que la signature du client doit être mise sur la 3ème page, **nb\_pages** sera à 1.

### LES DONNÉES JSON POUR LES RAPPORT / CERFA

* **code\_client** : code du client qui subit le rapport d’intervention
* **nom\_client** : nom du client qui subit le rapport d’intervention
* **email\_client** : mail du client qui subit le rapport d’intervention
* **telephone\_client** : numéros de téléphone du client qui subit le rapport d’intervention
* **portable\_client** : numéros de portable du client qui subit le rapport d’intervention
* **adresse\_facturation** : adresse de facturation
* **cp\_facturation** : code postale de facturation
* **ville\_facturation** : ville de facturation
* **lieu\_intervention** : lieu de l’intervention
* **adresse\_intervention** : adresse de l’intervention
* **cp\_intervention** : code postale de l’intervention
* **ville\_intervention** : ville de l’intervention
* **intervenant** : le technicien qui intervient sur l’intervention
* **description** : une description
* **intervention\_realisable** : si l’intervention est réalisable, oui ou null
* **equipier** : le nom de l’équipier s’il y en a un
* **compte\_rendu** : un petit compte rendu de l’intervention
* **materiel** : le matériel utilisé
* **intervention\_suite** : si un intervenant doit revenir sur l’intervention, oui ou null
* **facturable** : si l’intervention est déjà facturable, oui ou null
* **terminee** : si l’intervention est terminée, oui ou null
* **absent** : si le client était absent, oui ou null
* **fait-le** : date où a été fait le rapport d’intervention
* **devis\_a\_faire** : si le devis est à faire, oui ou null
* **photo\_avant** : chemin relatif photo avant intervention, depuis le dossier public de storage
* **photo\_apres** : chemin relatif photo après intervention, depuis le dossier public de storage
* **complements** : expliquer dans EXPLICATION--LAYOUT
* **complement\_client** : photo avec commentaire, maximum 2
* **signature** : image png, transformer en base 64

### LES DONNÉES JSON POUR LES TOKENS

dataToken garde les informations utiles au token, c’est une liste JSON avec 3 entrées :

* **client** : le nom du client / entreprise
* **document** : le type de document
* **uid** : le nom du devis ou rapport d’intervention / cerf

a

# **SÉCURITÉ & AUTHENTIFICATION**

IAPEL n’utilise pas de compte client ou de moyen qui y ressemble, donc pour vérifier que l’utilisateur n’accède pas au fichier des autres clients, les liens sont anonymisés grâce à des tokens. De plus, pour certains liens, il faut rajouter dans le header de la requête le mot de passe donc la clé doit être « secret-token », c’est le mot de passe défini dans le fichier .env .

## TOKEN

Le token est une chaîne de 60 caractères aléatoires constituée de minuscules, majuscules, chiffres, caractères spéciaux exclus. Il fonctionne grâce à la table tokens située dans la BDD, lors d’un appel d’une route avec un token, un middleware va vérifier si le token existe et s’il est valide. Le tokens est relié au document grâce à son chemin relatif qui débute depuis iapel/storage/app/public. Le token est utilisé pour les devis et pour les rapports d’intervention, la colonne used de la table n’est utilisée que pour les devis et est ignorée par les rapports d’intervention.

## HEADER

Pour certaines routes, une balise header dans la requête est demandée, il faut mettre une balise « secret-token » avec le mot de passe dans le header de la requête. Les tokens d’identification des documents ne sont pas la même chose que le secret-token qui lui permet d’ajouter une sécurité pour certaine route.

Les route avec header sont données dans la partie ROUTES .

## MIDDLEWARE & ACCÈS

Il existe dans IAPEL 5 middlewares dont un qui est créé dynamiquement dans api.php. Il y a 3 middlewares pour la sécurité des tokens et du secret-token (VerifHeaderPassword, VerifTokenMiddleware et VerifTokenWithHeaderMiddleware) et 2 middlewares pour l’anti-spam / bruteforce (AntiSpamMiddleware et throttle:anti-bruteforce dans api.php et définit dans web.php).

* **VerifHeaderPassword** : Vérifier le mot de passe dans le header de la requête,
* **VerifTokenMiddleware** : Vérifie la validité du token demandé (date d’expiration, utilisé, existe)
* **VerifTokenWithHeaderMiddleware** : Vérifie le mot de passe dans le header de la requête et vérifie la validité du token demandé. C’est un mixte des deux autres.
* **AntiSpamMiddleware** : N’est pas utilisé car remplacé pour l’instant par throttle:anti-bruteforce. AntiSpamMiddleware utilise l’URL et l’IP de l’utilisateur pour mettre en place un principe d’anti-spam. 5 requêtes par minute maximum pour une URL, modifiable.
* **throttle:anti-bruteforce** : Utilise des outils déjà inclus dans Laravel avec RateLimiter. 20 requêtes par minute maximum pour toutes les URL, modifiable.

# **FONCTIONNALITÉS**

## RESSEMBLANCE MODULE DEVIS & RAPPORT D’INTERVENTION

Le système de tokens entre la partie devis et rapport d’intervention / cerfa est le même pour tout le projet. Les seuls changements sont le type de documents, le chemin relatif et le champ used qui est utilisé uniquement par les devis.

## MODULE DEVIS

La génération du devis, ou plutôt l’import du devis, se fait via la route /api/ generate-token-devis qui récupère les informations contenues dans la requête et les enregistre dans un fichier JSON, juste après le token est créer et stocker en base de données. Le serveur renvoie en cas de réussite un message de succès, le token et l’url de signature, et en cas d’erreur un message d’erreur et un statut 500.

Après la génération du devis, APEL recherche le modèle du devis avec ses spécificités et renomme le modèle avec le nom du token, puis envoie le modèle de PDF du devis via l’url /devis/upload-pdf. Sur IAPEL, une fonction vérifie que le nom du PDF, donc le token, existe bien dans la base de données, puis ajoute le fichier dans le bon répertoire client.

L'accès au devis pour le signer électroniquement se fait via l'URL crée lors de la génération / import du devis. L’url est envoyée par mail au client depuis APEL. Cette URL utilise le token généré et ressemble normalement à ceci /devis/signature/token (méthode GET).

Après avoir signé et cliqué sur « signer électroniquement », la vue va envoyer sur l’url /devis/signature/token en méthode POST la signature et le token. Le serveur va mettre à jour le JSON et créer le PDF du devis signé et un autre PDF du devis signé et certifié. Pour certifier le devis, on utilise un script Python qui récupère le certificat et l’ajoute au PDF dans les Metadata.

Le devis ne peut être signé qu’une fois et lors d’une nouvelle visite avec le même token, une autre page sera montrée disant que le devis a bien été signé électroniquement.

Pour supprimer un devis, on passe par APEL et une requête en envoyer sur l’url /devis/delete-devis/token. IAPEL supprime tous les fichiers du répertoire, le répertoire, le token de base de données et renvoie un message de succès ou d’erreur en cas d’erreur.

Pour télécharger un devis, il faut avoir signé le devis depuis la page de signature. Après l'avoir signé, la page se recharge et permet au client de télécharger le devis. Le lien qui est utilisé pour télécharger le document depuis la vue est /devis/download-devis/token.

Une autre URL : /devis/check-devis/token permet de savoir si un devis certifié ou signé existe et renvoie true ou false.

## MODULE RAPPORT D’INTERVENTION

Le génération / import du rapport d’intervention / cerfa est logiquement la même que celle du devis avec une requête POST sur une route /api/generate-token-rapport-cerfa avec dans la requête (body) les données. Les données sont enregistrées dans le fichier JSON avec les données du token. Après le token est créé et enregistré dans la base de données. IAPEL renvoie le token, l’url vers le rapport d’intervention (bi) et un message de réussite, sinon il renvoie une erreur. La seule différence c’est le type de document et les données enregistrées dans le fichier JSON.

Pour remplir le bon d’intervention, il faut aller sur APEL MOBILE et cliquer sur le rapport d’intervention qui est apparu. La routes est /rapport-cerfa/bi/token. APEL MOBILE crée automatiquement le lien vers le bon d'intervention (bi). Le page bon d’intervention est répartie en 5 parties :

1. Détails de l’intervention, reprend toutes les informations envoyées lors de la création du token et les affiche.
2. Réalisation, affiche de nouveaux inputs au technicien comme un compte-rendu ou le matériel consommé.
3. Complément, permet d’ajouter 2 photos maximum avec un commentaire à chaque fois. Les photos sont envoyées sur le serveur grâce à la route /updload-visuel, on peut les supprimer grâce à la route /delete-visuel.
4. Particularité, cette partie est particulière car elle n’est pas affichée chez tous les clients mais que les clients qui ont un layout enregistré dans la base de données. Le layout est inclus grâce à un **@include**, le programme test avec un **@if** si un layout existe. **Voire Explication – LAYOUT. Voire Blade PHP.**
5. Signature, permet de signer le rapport d’intervention.

Lorsque le client a fini de rajouter les informations manquantes et clique sur le bouton « valider l’intervention et générer le PDF », la route /rapport-cerfa/submit est appelée. Les informations du token sont récupérées et sont ajoutées dans le fichier JSON du document. La route /rapport-cerfa/pdf/token est appelée.

Le pdfController est appelé car la route /rapport-cerfa/pdf/token est appelée, le pdfController (fonction show) va appeler lui la vue pdf (**pdf.blade.php**).

Ici ça risque d'être compliqué à comprendre, la génération du PDF se fait via une requête envoyée depuis la vue. Lorsque la vue est envoyée au côté client, blade va vérifier si le client est sur Android ou PC et va par rapport à ça créer une variable. L’url du PDF est créée au même moment grâce aux données envoyées par le contrôleur. Et lorsque le document va charger, une requête va être envoyée au serveur (/generate/generate-rapport\_intervention) pour générer le PDF du rapport d’intervention avec des paramètres (**client, document, uid, isAndroid**).

La génération du PDF se fait grâce au token et aux données stockées dans le fichier JSON relié au token. Un modèle de PDF existe dans le dossier rapport\_intervention qui va être utilisé pendant la génération du PDF. Les données seront enregistrées dans le PDF via des coordonnées X, Y et des tailles de texte. Les coordonnées sont écrites en dur dans le code. Lorsque le PDF a fini de se générer et d'enregistrer, le serveur renvoie le PDF.

Coté client, la réponse est reçue et affiche un message de réussite ou d’erreur. L'URL créée plus tôt va, lors de la fin du chargement, afficher le PDF car il vient d'être créé et l'URL existe.

Pour que APEL MOBILE puisse savoir quel token est reliée à quel document, la route /api/getToken existe et prend en paramètre URL le client, le document, l’uid et un header « secret-token » avec mot de passe. La route renvoie un JSON avec le token.

Ou une autre route /rapport-cerfa/list qui permet-elle de lister tous les documents d’un client. La route prend en paramètre URL le client et un header « secret-token » avec mot de passe. Renvoie un JSON avec le chemin vers le document (***paths***), le token (***token\_rapport***), s'il est valide ou non (***status***) et des données (***data***).

Pour supprimer un document, il faut passer par APEL MOBILE et appuyer sur le bouton supprimer à côté du rapport d’intervention. La route appelée est /delete/token et demande un header secret-token. La fonction supprime les documents, le dossier et le token de BDD.

Pour télécharger le devis pareil, il faut passer par APEL MOBILE et appuyer sur le devis. La route appelée est /rapport-cerfa/download/token et renvoie un téléchargement du PDF demandé.

Pour savoir si un rapport d’intervention / cerfa existe, la route /rapport-cerfa/check/token a été créé. Il renvoie true si le document existe et false sinon.

# **ROUTES**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **API** | | | | |
| **CONTROLLER -- FONCTION** | **URL** | **HEADER** | **METHODE** |  |
| TokenController -- generateDevis | /api/generate-token-devis | *SECRET-TOKEN* | POST |  |
| TokenController --  generateOther | /api/generate-token-rapport-cerfa | *SECRET-TOKEN* | POST |  |
| TokenController -- validateToken | /api/validate-token/{token} |  | GET |  |
| TokenController --  getToken | /api/getToken/{client}/{document}/{uid} | *SECRET-TOKEN* | GET |  |
| Aucun -- Imbriquer | /api/token | *SECRET-TOKEN* | GET |  |
|  |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **WEB** | | | | |
| **CONTROLLER -- FONCTION** | **URL** | **HEADER** | **METHODE** |  |
| Redirection | / |  | GET |  |
| Aucun – Imbriquer | /upload-visuel |  | POST |  |
| Aucun -- Imbriquer | /delete-visuel |  | POST |  |
| BiController -- delete | /delete/{token} | *SECRET-TOKEN* | GET |  |
|  |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **GENERATE** | | | | |
| **CONTROLLER -- FONCTION** | **URL** | **HEADER** | **METHODE** |  |
| PdfController -- generateCerfa | /generate/generate-cerfa\_15497 |  | GET |  |
| PdfController -- generateCerfa | /generate/generate-cerfa\_15497\_1 |  | GET |  |
| PdfController -- generateCerfa | /generate/generate-cerfa\_15497\_2 |  | GET |  |
| PdfController -- generateAttestationTVA | /generate/generate-cerfa\_13948-03 |  | GET |  |
| PdfController -- generateBi | /generate/generate-rapport\_intervention |  | GET |  |
| PdfController -- generateAttestationTVA | /generate/generate-cerfa\_13948-03 |  | POST |  |
| PdfController -- generateDownloadPDF | /generate/generate-download-pdf |  | POST |  |
|  |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **DEVIS** | | | | |
| **CONTROLLER -- FONCTION** | **URL** | **HEADER** | **METHODE** |  |
| DevisController --  Delete | /devis/delete-devis/{token} | *SECRET-TOKEN* | GET |  |
| PdfController --  viewDevis | /devis/pdf-devis/{token} |  | GET |  |
| DevisController -- check | /devis/check-devis/{token} | *SECRET-TOKEN* | GET |  |
| SignatureController -- show | /devis/signature/{token} |  | GET |  |
| SignatureController --  sign | /devis/signature/{token} |  | POST |  |
| DevisController -- download\_devis | /devis/download-devis/{token} |  | GET |  |
| DevisController -- download-devis-certifie | /download-devis-certifie/{token} |  | GET |  |
| PdfController -- upload | /devis/upload-pdf |  | POST |  |
|  |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **RAPPORT D’INTERVENTION** | | | | |
| **CONTROLLER -- FONCTION** | **URL** | **HEADER** | **METHODE** |  |
| BiController -- show | /rapport-cerfa/bi/{token} |  | GET |  |
| BiController -- submit | /rapport-cerfa/submit/{token} |  | POST |  |
| PdfController -- show | /rapport-cerfa//pdf/{token} |  | GET |  |
| BiController -- open | /rapport-cerfa/open/{token} | *SECRET-TOKEN* | GET |  |
| BiController -- getDocuments | /rapport-cerfa/documents/{client} |  | GET |  |
| BiController -- download | /rapport-cerfa/download/{token} |  | GET |  |
| BiController -- check | /rapport-cerfa/check/{token} | *SECRET-TOKEN* | GET |  |
| BiController -- listSavedDocs | /rapport-cerfa/list/{clien  t} | *SECRET-TOKEN* | GET |  |
|  |  |  |  |  |