

GUIDE OFFICIEL · CLIENT

Installer et exploiter votre plateforme MutualBoost de A à Z.

Documentation CLIENT approfondie : prérequis, hébergement, activation, gestion de la mutuelle, Flutter, OneSignal/FCM/APNs, paiements, supervision, mises à jour signées et restauration.

Édition du 18 octobre 2026

28 chapitres pratiques

Domaines illustratifs : mondomaine.com / licences.mondomaine.com

Suivez ce parcours avec les accès fournis par votre éditeur. La licence couvre votre installation ; vos adhérents se connectent ensuite avec leur compte habituel.

Sommaire

01. Comprendre les deux distributions
02. Checklist d'installation avant toute commande
03. Avant de commencer
04. Prérequis techniques
05. Installer le serveur Laravel
06. Base de données, migrations et initialisation
07. Hébergement HTTPS, stockage et cron
08. Administrateurs, droits et première connexion
09. Activer la licence de votre installation
10. Licences internes, Envato et activation CLIENT
11. Centre de mises à jour signées
12. Hébergement et tâches automatiques
13. OneSignal : notifications Android et iOS
14. Email SMTP, SMS et modèles de messages
15. Paiements : activation et rapprochement
16. Installer et connecter Flutter
17. Flutter : projet complet, API et identité
18. Compiler et publier Android
19. Compiler et publier iOS
20. Publication Android, iOS et confidentialité
21. Paramètres de la mutuelle et gestion quotidienne
22. Apparences, langues et contenu de vitrine
23. Configurer le site et l'application
24. Paiements, notifications et téléchargements
25. Mises à jour et sauvegardes
26. Sécurité, journalisation et sauvegardes
27. Dépannage et réponses aux incidents
28. Dépannage rapide

01 Comprendre les deux distributions

Distinguer le serveur éditeur de la plateforme installée chez chaque mutuelle.

01.01 ÉDITEUR : rôle exclusif de GBAMA CODES

La distribution ÉDITEUR pilote l'émission de licences, les achats Envato, le portefeuille des installations et les publications logicielles signées. Elle s'installe sur une instance Laravel dédiée comme `licences.mondomaine.com` et protège ses clés privées hors du répertoire public.

01.02 CLIENT : rôle de chaque mutuelle

Chaque mutuelle dispose de sa propre installation Laravel (`mondomaine.com`), de sa base de données et de sa configuration. Le Web et Flutter consomment ce serveur unique ; une licence logicielle protège l'installation, pas chaque adhérent.

01.03 Distinguer sources cumulatives et projets autonomes

Les ZIP de cette livraison regroupent des fichiers applicatifs à fusionner avec un Laravel complet. Il leur manque le squelette verrouillé (`artisan`, `composer.json`, `composer.lock`) ; Flutter lib/ n'est pas un projet Android/iOS autonome. Ne lancez pas Composer dans un dossier sans `composer.json`.

01.04 Séparer les environnements

Prévoir production, préproduction et développement avec domaines, bases, sauvegardes et identifiants distincts. Ne jamais connecter une application de test aux vrais moyens de paiement ni réutiliser la clé privée ÉDITEUR sur un CLIENT.

02 Checklist d'installation avant toute commande

Prévenir les erreurs de Composer, de permissions et de déploiement déjà rencontrées.

02.01 Confirmer les fichiers de base

Entrez dans le répertoire qui contient `artisan` ET `composer.json` ET `composer.lock`. Sur une archive cumulative seule, arrêtez-vous et récupérez le projet Laravel complet d'origine.

```
cd /home/UTILISATEUR/mondomaine.com
pwd
for f in artisan composer.json composer.lock bootstrap/app.php; do test -f "$f" && echo
    "OK $f" || echo "ABSENT $f"; done
php -v; composer --version
```

02.02 Contrôler la compatibilité PHP

Laravel 13 exige PHP 8.3 ou une version supérieure supportée par les packages verrouillés. Vérifiez les extensions PHP réellement utilisées : `pdo_mysql`, `cURL`, `DOM`, `Fileinfo`, `OpenSSL`, `Sodium`, `ZIP`, `Mbstring` et `XML`. Le même binaire PHP doit être utilisé par Artisan et Composer.

```
php -m | grep -Ei "pdo|curl|dom|fileinfo|openssl|sodium|zip|mbstring"  
composer check-platform-reqs --no-dev
```

02.03 Contrôler la racine Web et les droits

La racine du domaine HTTPS doit être /home/UTILISATEUR/mondomaine.com/public. Vérifiez le certificat TLS, les permissions 755 pour les dossiers publics et 644 pour les fichiers ; storage et bootstrap/cache doivent être accessibles en écriture au service PHP.

```
namei -l /home/UTILISATEUR/mondomaine.com/public/.htaccess  
ls -ld public storage bootstrap/cache
```

02.04 Documenter les choix de déploiement

Avant la fusion, notez la branche/version sources, le rôle CLIENT ou ÉDITEUR, le nom du domaine canonique, le fournisseur d'hébergement, le chemin de sauvegarde privé et les moyens de paiement/notifications effectivement activés.

03 Avant de commencer

Identifiez la bonne livraison et préparez un environnement de travail.

03.01 Distinguer une installation et une mise à jour

Une installation neuve exige un projet Laravel complet avec artisan, composer.json, composer.lock et public/, ainsi qu'un projet Flutter complet avec pubspec.yaml, android/ et ios/. Une archive cumulative ou un dossier lib seul se fusionne avec le projet d'origine ; il ne remplace pas les fichiers de structure manquants.

03.02 Préparer les accès

Prévoyez un domaine HTTPS, un accès aux fichiers et au terminal de l'hébergement, une base de données dédiée et les sources complètes. Le serveur Laravel doit fonctionner avant de brancher l'application. Pour une mise en place soumise à licence, demandez aussi à l'éditeur l'adresse de sa centrale, sa clé publique et votre clé d'activation.

03.03 Choisir un environnement de test

Travaillez d'abord sur un sous-domaine et une base séparée. Pour un site déjà installé, passez directement au chapitre Mises à jour et sauvegardes.

04 Prérequis techniques

Vérifiez le serveur et les outils locaux avant d'installer les dépendances.

04.01 Serveur Laravel

Laravel 13 nécessite PHP 8.3 ou supérieur. Le fichier composer.lock et composer check-platform-reqs confirment les exigences exactes de la livraison. Vérifiez notamment cURL, DOM/XML, Fileinfo, Mbstring, OpenSSL et PDO avec le pilote de votre base. Le module de licence requiert aussi PHP Sodium.

```
php -v
php -m
composer --version
```

04.02 Poste Flutter

Installez le SDK Flutter compatible avec les contraintes du pubspec.yaml, Git et les outils Android. Pour iOS, utilisez un Mac avec Xcode. Conservez le pubspec.lock et les fichiers Gradle de votre projet ; ne choisissez pas leurs versions au hasard.

```
flutter --version
flutter doctor -v
```

05 Installer le serveur Laravel

Ce parcours concerne exclusivement une première installation sur une base vide.

05.01 1. Vérifier le projet complet et installer Composer

Ouvrez d'abord le projet Laravel complet d'origine, en local ou sur le serveur. Les archives cumulatives CLIENT/ÉDITEUR MutualBoost ne contiennent pas composer.json, composer.lock ni artisan : elles doivent être fusionnées avec le projet complet. Vérifiez ces trois fichiers AVANT Composer. Vous pouvez préparer les dépendances en local avec la bonne version de PHP, mais un vendor construit sous macOS ou Windows n'est pas garanti compatible avec le serveur Linux : privilégiez Composer sur o2switch ou une préparation sous Linux compatible. Sur le serveur, utilisez le même PHP pour Composer et Artisan.

```
cd /home/UTILISATEUR/mondomaine.com
pwd
for f in artisan composer.json composer.lock bootstrap/app.php; do test -f "$f" || { echo
  "STOP: missing $f"; exit 1; }; done
php -v
composer install --no-dev --prefer-dist --optimize-autoloader
composer check-platform-reqs --no-dev
```

Attention : Si Composer affiche « could not find a composer.json file », vous êtes dans le mauvais dossier ou n'avez que l'archive partielle. Ne faites pas composer init ni composer update : retrouvez le Laravel complet d'origine. Un dossier local est utile pour contrôler les fichiers, mais ne crée pas les fichiers manquants.

05.02 2. Configurer l'environnement

Créez .env à partir du modèle fourni uniquement si aucun fichier .env n'existe. Renseignez votre domaine et une base neuve. L'exemple ci-dessous contient des valeurs de remplacement ; il ne contient aucun identifiant de votre serveur.

```
APP_ENV=production
APP_DEBUG=false
APP_URL=https://mondomaine.com

DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=mualboost
DB_USERNAME=utilisateur_bdd
DB_PASSWORD="REPLACER_PAR_VOTRE_MOT_DE_PASSE"
```

05.03 3. Générer la clé et créer les tables

Sur cette nouvelle installation uniquement, générez APP_KEY une fois, vérifiez la connexion à la base et appliquez les migrations. Gardez cette clé lors des mises à jour.

```
php artisan key:generate
php artisan migrate --force
php artisan migrate:status
```

Attention : Sur un serveur existant, ne régénérez pas APP_KEY et ne lancez pas de commande qui recrée les tables.

05.04 4. Initialiser les données de la livraison

Sur une base neuve, le DatabaseSeeder fourni initialise les réglages et un jeu de démonstration. Il refuse une base déjà peuplée. Exécutez cette étape seulement si vous souhaitez cette initialisation.

```
php artisan db:seed --class=DatabaseSeeder --force
```

Attention : Cette initialisation crée des données et des comptes fictifs. Ce n'est pas une procédure de mise à jour.

05.05 5. Régler le dossier Web et les fichiers

Le domaine doit servir le dossier public/ du projet. Autorisez PHP à écrire dans storage/ et bootstrap/cache/. Si public/storage est absent, créez le lien prévu. Si un vrai dossier existe déjà à cet emplacement, conservez-le et ses fichiers.

```
if [ ! -e public/storage ] && [ ! -L public/storage ]; then
    php artisan storage:link
fi
```

05.06 6. Ouvrir la plateforme et sécuriser le compte

Ouvrez /login. Dans la livraison actuelle, le seeder écrit les identifiants initiaux dans `storage/app/mutualboost-demo-identifiants.txt`, hors du dossier public. Consultez ce fichier uniquement sur votre serveur. Modifiez ensuite les coordonnées et le mot de passe de l'administrateur, puis configurez l'identité de votre mutuelle.

```
php artisan config:clear
php artisan view:cache
```

06 Base de données, migrations et initialisation

Configurer MySQL sans effacer une installation existante.

06.01 Base neuve et identifiants dédiés

Créez une base MySQL et un utilisateur avec les privilèges strictement nécessaires. Conservez leurs identifiants dans `.env` hors public/ ; n'utilisez jamais la base d'un autre client.

```
APP_ENV=production
APP_DEBUG=false
APP_URL=https://mondomaine.com
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=mutualboost_prod
DB_USERNAME=utilisateur_db
DB_PASSWORD="REPLACER_SECRET"
```

06.02 Composer et premier bootstrap

Installez les dépendances issues de `composer.lock` puis validez le chargement Laravel. En cas d'erreur `package:discover`, diagnostiquez `bootstrap/app.php` et `bootstrap/cache` au lieu de relancer `composer update`.

```
composer install --no-dev --prefer-dist --optimize-autoloader
composer check-platform-reqs --no-dev
php artisan about
```

06.03 Migrations consolidées et historique

Sur une base vide, exécutez `migrate --force` puis vérifiez `migrate:status`. Sur une base existante, sauvegardez d'abord : ne lancez jamais `migrate:fresh`, `db:wipe` ni un nettoyage manuel de la table migrations. Des migrations historiques peuvent avoir été archivées après consolidation.

```
php artisan migrate:status
php artisan migrate --force
php artisan migrate:status
```

06.04 Seeders : seulement avec la bonne procédure

DatabaseSeeder initialise les pages, paramètres et la démonstration sur une installation neuve. Il peut refuser une base peuplée ; ne le forcez pas en production. Le DemoMutualSeeder attend un contexte préparé par DatabaseSeeder et peut nécessiter Faker dans les dépendances installées.

```
php artisan db:seed --class=DatabaseSeeder --force
```

Attention : Ne lancez pas DemoMutualSeeder isolément et ne publiez jamais des comptes de démonstration sur une vraie mutuelle.

06.05 Clé de chiffrement APP_KEY

Générez APP_KEY une seule fois pour une nouvelle installation. Elle protège les données chiffrées : toute régénération sur un site existant peut rendre les données antérieures illisibles.

```
php artisan key:generate # NOUVELLE INSTALLATION UNIQUEMENT
```

07 Hébergement HTTPS, stockage et cron

Racine Web, fichiers, tâches planifiées et supervision.

07.01 Dossier public et lien de stockage

Le serveur doit servir public/index.php. Si public/storage n'existe pas, utilisez php artisan storage:link ; s'il est déjà un vrai répertoire contenant des fichiers, ne le supprimez pas sans inventaire et sauvegarde.

```
cd /home/UTILISATEUR/mondomaine.com
if [ ! -e public/storage ] && [ ! -L public/storage ]; then php artisan storage:link; fi
```

07.02 Cron Laravel : une seule entrée par installation

Dans cPanel ou crontab, créez UNE tâche qui appelle schedule:run chaque minute avec le PHP de l'installation. schedule:list affiche les événements prévus mais ne prouve pas qu'un cron système est présent.

```
* * * * * cd /home/UTILISATEUR/mondomaine.com && /CHEMIN/PHP artisan schedule:run >>
/home/UTILISATEUR/logs/mutualboost-scheduler.log 2>&1
```

07.03 Inventaire des tâches planifiées

Contrôlez les rappels de cotisations, génération des dettes, rapprochement des paiements, expiration de contrats, notifications automatiques, rafraîchissement des licences et contrôle quotidien des versions. Les périodicités exactes doivent être lues dans schedule:list sur le serveur installé.

```
php artisan schedule:list
crontab -l | grep -F mondomaine.com
php artisan licenses:diagnose --online # CLIENT
```

07.04 Queues et mail : selon votre configuration

Si QUEUE_CONNECTION utilise database/redis, prévoyez un worker supervisé ou une méthode d'exécution compatible avec l'hébergement. Si votre installation utilise sync, aucune file asynchrone n'est démarrée ; vérifiez l'impact sur les notifications et les temps de réponse.

```
php artisan queue:failed
php artisan queue:work --once
```

Attention : Exécuter un worker persistant uniquement si l'hébergement et la configuration le permettent.

07.05 HTTPS, sauvegardes et journaux

Activez TLS, redirection HTTP→HTTPS, APP_DEBUG=false, logs privés et sauvegardes base + fichiers. Testez périodiquement une restauration sur préproduction. Ne publiez jamais .env, storage/app/private, logs ou dumps SQL.

08 Administrateurs, droits et première connexion

Comptes initiaux, sécurité, rôles et parcours public.

08.01 Créer un compte administrateur autorisé

L'accès /login doit fonctionner même avant activation de licence afin que le véritable administrateur puisse rejoindre /admin/license. Pour l'installation de démonstration uniquement, DefaultAdminSeeder peut fournir un compte initial ; changez immédiatement son mot de passe et désactivez les accès de test avant la production.

```
php artisan db:seed --class=DefaultAdminSeeder --force
```

Attention : Ne publiez pas les identifiants prédéfinis dans une page de documentation publique.

08.02 Rôles et capacités

Créez des rôles nommés (secrétariat, économe, responsable, gestionnaire) et attribuez les capacités strictement nécessaires. Réservez la licence, les clés de paiement, les mises à jour et l'administration des rôles au personnel habilité.

08.03 Navigation intelligente du site

Dans Administration → Site et application → Boutons Connexion/Inscription/Don, paramétrez les actions publiques. Le menu change selon Visiteur, Adhérent ou Administrateur ; les inscriptions doivent aussi respecter la politique d'ouverture et les restrictions de licence.

08.04 Désactiver les comptes démo

Les identifiants de la mutuelle fictive doivent être réservés aux environnements isolés. Vérifiez les droits, l'absence de fonctions financières réelles en mode démo et la déconnexion sécurisée via POST/CSRF.

09 Activer la licence de votre installation

Votre mutuelle utilise le rôle client. L'éditeur vous communique les informations de sa centrale.

09.01 1. Recevoir le bon dossier et les informations de licence

Votre installation réunit le projet Laravel complet, ses fichiers de mise à jour compatibles et, si vous avez l'application mobile, le projet Flutter complet. Recevez de l'éditeur l'URL HTTPS finale de sa centrale, sa clé publique et votre clé interne ou votre code d'achat CodeCanyon. Une archive cumulative ne remplace pas les fichiers de structure absents.

Attention : Le rôle client correspond à la plateforme de votre mutuelle. Les adhérents utilisent ensuite votre site et votre application ; ils n'activent pas chacun une licence logicielle.

09.02 2. Configurer uniquement le rôle client

Dans le fichier .env du serveur Laravel de votre mutuelle, renseignez votre domaine et les informations publiques fournies par l'éditeur. L'exemple n'est pas une centrale déjà disponible. Après modification, videz le cache de configuration.

```
APP_URL=https://mondomaine.com
MUTUALBOOST_LICENSE_ROLE=client
MUTUALBOOST_LICENSE_SERVER_URL=https://licences.mondomaine.com
MUTUALBOOST_LICENSE_PUBLIC_KEY="CLE_PUBLIQUE_FOURNIE_PAR_EDITEUR"
MUTUALBOOST_LICENSE_REQUIRED=true
```

Attention : Après avoir enregistré .env, exécutez php artisan config:clear dans le terminal du projet Laravel. Le jeton auteur Envato et la clé privée de signature restent chez l'éditeur ; ils ne sont nécessaires ni sur votre serveur ni dans Flutter.

09.03 3. Saisir votre clé dans l'administration

Connectez-vous à /admin/license : Administration → Licence de la plateforme. Choisissez Licence interne MutualBoost si l'éditeur vous a remis une clé MBI, ou CodeCanyon / Envato si vous disposez d'un code d'achat. Saisissez-le et cliquez sur Activer la licence. Vérifiez ensuite le domaine et l'état Licence active. Un compte adhérent ou ADMIN-DEMO ne peut pas effectuer cette activation.

Attention : Le code d'achat se trouve dans vos téléchargements Envato, avec le certificat de licence. Le mot de passe de votre compte Envato n'est pas demandé.

09.04 4. Vérifier le renouvellement et la continuité de service

Conservez le cron Laravel sur votre hébergement. Le serveur vérifie localement l'attestation pendant la navigation et tente son renouvellement lorsqu'elle atteint 24 heures. Une coupure temporaire laisse fonctionner l'attestation encore valide, jusqu'à sept jours et dans la limite d'une échéance interne. Vérifier maintenant permet de relancer le contrôle après résolution d'un problème.

```
php artisan schedule:list  
php artisan licenses:refresh --force
```

Attention : La date de fin du support Envato est présentée séparément du droit d'utilisation. Le module ne transforme pas cette date de support en expiration de licence.

09.05 5. Transférer votre plateforme vers un autre domaine

Avant le transfert, ouvrez Transférer ou remplacer la licence dans /admin/license, confirmez la désactivation, puis activez sur le nouveau serveur et son domaine canonique. Gardez la même identité privée lors d'une simple mise à jour sur le même domaine. Si l'ancien serveur est inaccessible, demandez à l'éditeur de libérer l'installation.

Attention : Ne copiez pas une installation activée pour ouvrir une deuxième plateforme. Le nombre de places dépend de votre licence ; un achat CodeCanyon est géré avec une installation active par produit final.

09.06 6. Comprendre une restriction et contacter l'assistance

Une attestation dépassée, une échéance interne ou une suspension peut interrompre les nouvelles écritures Web et API avec HTTP 423. La consultation, la récupération du compte, l'accès administrateur à la licence et les confirmations de paiements déjà initiés restent disponibles. Transmettez à l'éditeur le domaine, le message affiché et l'heure du problème, sans publier votre clé complète, .env ou vos mots de passe.

Attention : Aucune centrale réelle ni aucun achat Envato ne sont activés par la simple lecture de ce guide. Les informations de raccordement viennent de votre éditeur.

10 Licences internes, Envato et activation CLIENT

Comprendre la durée, les attestations et les contrôles sans bloquer l'activation.

10.01 Activer directement depuis Laravel CLIENT

Dans /admin/license, un administrateur autorisé saisit la clé interne ou le code d'achat Envato. Le serveur contacte la centrale HTTPS, épingle sa clé publique lors du premier échange de confiance et vérifie l'attestation signée. Ne saisissez jamais la clé privée de signature dans l'interface CLIENT.

```
APP_URL=https://mondomaine.com  
MUTUALBOOST_LICENSE_SERVER_URL=https://licences.mondomaine.com  
MUTUALBOOST_LICENSE_ROLE=client
```

10.02 Comprendre les durées

Une licence interne sans date de fin choisie est illimitée ; les droits d'usage Envato sont distincts du support. L'attestation technique a une durée limitée (jusqu'à sept jours) et est renouvelée périodiquement : cette date de renouvellement ne représente pas la date d'expiration commerciale.

10.03 Diagnostiquer sans exposer les secrets

Vérifiez le domaine APP_URL, l'empreinte publiée, la clé publique épinglée, le certificat et l'identité d'installation. Une empreinte n'est pas une clé publique complète. N'effacez pas client.json ou authority.json pour forcer une activation.

```
php artisan config:clear
php artisan licenses:diagnose --online
php artisan licenses:refresh
```

10.04 Suspension et transfert

La suspension/révocation se pilote depuis la centrale et n'efface pas les opérations historiques. Pour changer réellement de domaine, préparez les sauvegardes et utilisez la libération/transfert de l'installation plutôt que de copier silencieusement un certificat sur un nouveau serveur.

11 Centre de mises à jour signées

Vérifier, sauvegarder, installer et restaurer sans prise de contrôle par le navigateur.

11.01 Installer le module d'updates une seule fois

La première intégration se fait par transfert fiable des sources, pas via une auto-mise à jour non encore installée. Vérifiez Sodium et ZIP. La clé publique d'updates est indépendante de la licence et doit être épinglée dans MUTUALBOOST_UPDATES_PUBLIC_KEY sur CLIENT.

```
php -m | grep -Ei "sodium|zip"
php artisan route:list --path=admin/updates
```

11.02 Contrôler avant téléchargement

La page /admin/updates affiche la version et les nouveautés. updates:check compare des manifestes signés. Un CLIENT actif peut télécharger via updates:download ; la signature Ed25519, la distribution ciblée et SHA-256 sont vérifiés avant copie.

```
php artisan updates:check
php artisan updates:download # CLIENT
```

11.03 Installer depuis SSH avec sauvegarde

L'installation des fichiers se lance uniquement avec une confirmation SSH et une version publiée. Sauvegardez base et fichiers. Pour les releases avec migration, fournissez le chemin du dump SQL, exécutez ensuite migrate --force, contrôlez les données et finalisez par updates:complete.

```
php artisan updates:apply VERSION --confirm=VERSION
--db-backup=/home/UTILISATEUR/sauvegardes/mondomaine-avant.sql
php artisan migrate --force # SI RELEASE AVEC MIGRATIONS
php artisan updates:complete --confirm=VERSION # SI MAINTENANCE EN ATTENTE
```

11.04 Récupération et limites du rollback

Si le code déployé échoue, utilisez `updates:rollback --confirm=VERSION` et vérifiez le service. Le rollback applicatif ne restaure NI les tables modifiées, NI les nouvelles dépendances Composer. Pour un retour arrière SQL, restaurez séparément une sauvegarde testée.

```
php artisan updates:rollback --confirm=VERSION
php artisan optimize:clear
```

11.05 Cron de surveillance, pas d'installation silencieuse

`updates:check` est planifié quotidiennement dans `routes/console.php`. Le cron ne doit ni appeler `updates:apply` ni accepter un ZIP arbitraire fourni par formulaire Web. Conservez un journal de version, auteur, notes, contrôle SHA et restauration.

12 Hébergement et tâches automatiques

Préparez le domaine, le planificateur et, si nécessaire, les files de traitement.

12.01 cPanel et o2switch

Sélectionnez un PHP compatible pour le domaine et pour le terminal. Configurez la racine du domaine sur `public/`, activez HTTPS et créez la base ainsi que son utilisateur avec les droits adaptés. Si vous utilisez un chemin PHP particulier, remplacez `php` par ce chemin dans les commandes.

12.02 Planificateur Laravel

Ajoutez une seule entrée cron exécutée chaque minute. Remplacez les deux chemins de l'exemple. Consultez `schedule:list` pour vérifier que les tâches de la livraison sont effectivement enregistrées. Elles couvrent notamment le rapprochement des paiements, les échéances de contrats et les rappels.

```
* * * * * cd /chemin/vers/mutualboost && /chemin/vers/php artisan schedule:run >>
storage/logs/scheduler.log 2>&1
```

Attention : Contrôle : `php artisan schedule:list`. Le fuseau horaire du projet détermine l'heure d'exécution des tâches.

12.03 Files de traitement

Si votre configuration utilise une file asynchrone, prévoyez un processus `queue:work` supervisé et relancé par votre hébergeur. Une configuration synchrone n'exige pas ce processus. Vérifiez le mode de votre livraison avant de l'activer.

13 OneSignal : notifications Android et iOS

Application OneSignal, FCM, APNs, secrets côté Laravel et réception Flutter.

13.01 Créer votre application OneSignal

Depuis le tableau de bord OneSignal, créez une application pour votre organisation. Dans Settings → Keys & IDs, récupérez App ID (UUID, partageable avec le SDK) et REST API Key (SECRET, serveur uniquement). Utilisez des identifiants distincts entre test et production.

13.02 Configurer Android et Firebase Cloud Messaging

Dans Firebase Console, créez/associez le projet Android avec le package exact de l'application, activez FCM et configurez les identifiants de service demandés par OneSignal. Ajoutez google-services.json seulement selon la configuration native prévue par le projet ; ne copiez jamais une clé de serveur dans Flutter.

13.03 Configurer Apple Push Notification service

Dans Apple Developer, utilisez le bon Bundle ID, activez Push Notifications et fournissez à OneSignal une clé APNs ou le certificat conforme à sa procédure. Configurez les capacités Xcode Push Notifications et Background Modes/Remote notifications si requises et testez sur un véritable iPhone.

13.04 Brancher Laravel et protéger la REST API Key

Dans Administration → Paramètres généraux → API / Notifications, renseignez onesignal_app_id, onesignal_rest_api_key et activez onesignal_enabled seulement après validation. Laravel rejette les valeurs fictives ; un champ secret laissé vide ne doit pas écraser l'ancien secret. La clé REST ne doit jamais être rendue dans le HTML public ou dans l'API Flutter.

13.05 Initialiser le SDK Flutter et l'utilisateur

Conservez la dépendance onesignal_flutter du pubspec.yaml, initialisez le SDK avec l'App ID (non secret), demandez les permissions au moment approprié et associez l'utilisateur connecté à l'identifiant externe attendu par Laravel. À la déconnexion, dissociez l'identité conformément au SDK installé.

```
// Schéma d'intégration : adapter à la version onesignal_flutter du projet
// OneSignal.initialize("APP_ID_UUID");
// OneSignal.login("IDENTIFIANT_UTILISATEUR_LARAVEL");
```

13.06 Tester chaque chemin de notification

Tester sur deux appareils : permission, token/abonnement, message direct, cible utilisateur, rappel cotisation, ouverture profonde et déconnexion. Les contrôleurs Laravel fournis contiennent encore des appels OneSignal historiques (api/v1 + include_external_user_ids) ; vérifiez leur compatibilité avec les API OneSignal actuelles avant de considérer les pushes comme opérationnels.

```
php artisan schedule:list
php artisan notify:automated --help
```

13.07 Respecter les préférences et la confidentialité

Publiez les règles de confidentialité et collectez les consentements nécessaires. Ne transmettez pas de diagnostics médicaux ou détails financiers sensibles en texte de notification sur l'écran verrouillé. Vérifiez la déclaration de confidentialité des magasins mobiles.

14 Email SMTP, SMS et modèles de messages

Configurer les fournisseurs réels, les expéditeurs et les scénarios métier.

14.01 Email : SMTP opérationnel

Dans Paramètres généraux, section API, renseignez smtp_host, smtp_port, smtp_username, smtp_password, smtp_encryption, smtp_from_address et smtp_from_name. Le serveur doit pouvoir joindre ce port sortant. Vérifiez SPF/DKIM/DMARC chez votre fournisseur et envoyez un message test.

14.02 SMS : choisir le fournisseur

Le sélecteur prévu contient Twilio, Infobip, Orange SMS API ou un fournisseur personnalisé. Saisissez sms_provider, sms_api_key, sms_api_secret et sms_sender_id selon votre contrat ; testez la livraison vers un numéro autorisé avant d'activer des relances massives.

14.03 Personnaliser les événements

Activez par événement welcome, payment, approval, rejection, new_dependent, death_notice ou reminder les canaux notify_*_email, notify_*_sms et notify_*_push. Ajustez objets, textes et titres, puis testez une vraie exécution sur un compte de test.

14.04 Surveiller les incidents

Consultez les logs Laravel, les éventuels jobs en échec, les quotas fournisseurs, les refus d'authentification et les erreurs HTTP. N'enregistrez pas les REST API Keys en clair dans des rapports partagés.

15 Paiements : activation et rapprochement

Paramétrer les opérateurs sans confondre présentation commerciale et authentification API.

15.01 Choisir les intégrations disponibles

Le code comprend des réglages pour Wave, CinetPay, TouchPay, PayPal et Flutterwave ; d'autres paramètres Fedapay, Jeko et Stripe existent selon le parcours. Activez uniquement le fournisseur réellement raccordé à son API et à son contrat marchand.

15.02 Saisir les clés dans l'administration

Dans Paramètres généraux → Paiements, renseignez les identifiants adaptés : Wave API key/signing secret ; CinetPay Site ID/API key/secret ; TouchPay codes d'agence et authentification ; PayPal Client ID/secret/environnement ; Flutterwave secret. Les logos, noms publics et ordre d'affichage n'affectent pas les identifiants techniques.

15.03 Retours, webhooks et double paiement

Configurez les URLs de retour, annulation et webhook selon les routes réellement présentes et le tableau de bord opérateur. Validez côté serveur la signature/référence, le montant, la devise, le statut et l'idempotence ; ne considérez jamais le simple retour navigateur comme preuve de paiement.

15.04 Faire des essais complets en sandbox

Vérifiez paiements réussis, échoués, annulés, différés, webhooks répétés, pertes réseau et rapprochement payments:reconcile-pending. Inspectez les journaux sans publier numéros de carte, secrets ou données de santé.

```
php artisan payments:reconcile-pending --help
php artisan schedule:list
```

16 Installer et connecter Flutter

Commencez quand le serveur HTTPS et ses API sont disponibles.

16.01 1. Ouvrir le projet complet

Ouvrez le dossier contenant pubspec.yaml. Un correctif lib doit être fusionné dans ce projet. Installez les dépendances et examinez les diagnostics avant de compiler.

```
flutter pub get
flutter analyze
```

16.02 2. Renseigner le serveur de votre mutuelle

Dans les sources Flutter vérifiées, lib/services/api_service.dart définit baseUrl et storageUrl. Remplacez ces deux adresses par celles de votre serveur. Conservez /api pour l'API et /storage/ pour les fichiers publics. Une version différente peut centraliser ces valeurs ailleurs : recherchez leurs définitions.

```
static const String baseUrl = 'https://mondomaine.com/api';
static const String storageUrl = 'https://mondomaine.com/storage/';
```

Attention : Flutter se connecte à votre plateforme cliente. Ne remplacez pas baseUrl par l'adresse de la centrale de licences.

16.03 3. Régler l'identité de l'application

Vérifiez le nom affiché, les icônes, l'identifiant Android et le Bundle Identifier iOS dans les projets natifs fournis. Utilisez vos propres identifiants pour une nouvelle application. Les identifiants d'une application déjà publiée doivent être conservés.

16.04 4. Tester sur un appareil

Détectez votre appareil ou émulateur puis lancez l'application. Vérifiez l'accueil, la connexion, les médias, les langues et une consultation des cotisations avec des données de test.

```
flutter devices
flutter run
```

Attention : Le téléphone doit pouvoir joindre le domaine HTTPS. localhost depuis un téléphone désigne ce téléphone, pas votre ordinateur.

17 Flutter : projet complet, API et identité

Brancher les sources Dart au serveur Laravel et préserver les variantes de styles.

17.01 Rassembler le projet mobile complet

Travaillez avec le dépôt Flutter complet comprenant pubspec.yaml, pubspec.lock, lib/, android/, ios/ et les assets. Le ZIP lib/ cumulatif ne suffit pas pour compiler des binaires ; faites une sauvegarde et fusionnez ensuite la nouvelle lib/ dans le vrai projet.

17.02 Configurer les URLs de votre serveur

Utilisez les paramètres de compilation effectivement reconnus par le projet MutualBoost ; ne laissez pas une URL de démo codée en dur. Le domaine API et l'URL des médias doivent pointer vers la même instance CLIENT HTTPS.

```
flutter pub get
flutter run --dart-define=MUTUALBOOST_API_BASE_URL=https://mondomaine.com/api
--dart-define=MUTUALBOOST_STORAGE_URL=https://mondomaine.com/storage/
```

17.03 Connexion, rôle et licence partagés

Les jetons d'authentification, rôles et permissions proviennent de Laravel. Une activation effectuée dans /admin/license déverrouille site et application selon les droits. Flutter doit afficher les états de licence sans jamais intégrer les clés privées ou une validation cryptographique qui contournerait Laravel.

17.04 Cinq styles et messages d'état

Conservez les cinq modèles visuels et leurs couleurs dynamiques. Le bandeau de messages d'état se place juste sous chaque AppBar, défile horizontalement, interprète une ligne comme un message et disparaît si aucun message n'est défini. Les couleurs du texte/fond dépendent du statut et sont pilotées par Laravel.

17.05 Tester l'intégration complète

Sur un appareil Android et un iPhone, testez login, inscription autorisée, adhérent, rôles, licence inactive/active, notifications, affichage des médias, paiements sandbox, traduction et changement de style. Réexécutez flutter analyze après toute fusion.

```
flutter doctor -v
flutter analyze
flutter test
```

18 Compiler et publier Android

Préparez la signature de production avant de distribuer votre application.

18.01 Signature et configuration

Configurez votre clé de signature dans le projet Android selon la procédure Flutter officielle. Conservez une sauvegarde privée de cette clé et de ses mots de passe. Vérifiez aussi la version de l'application, les permissions et les réglages de notifications.

18.02 Générer les fichiers de diffusion

L'AAB sert à la publication Google Play ; l'APK permet des tests de distribution directe. Les sorties habituelles sont build/app/outputs/bundle/release/ et build/app/outputs/flutter-apk/. Testez le fichier release avant publication.

```
flutter build appbundle --release
flutter build apk --release
```

Attention : La compilation ne publie pas automatiquement l'application. Téléversez ensuite l'AAB dans votre propre compte Google Play Console et complétez sa fiche.

19 Compiler et publier iOS

La distribution iOS se prépare sur macOS avec le projet Xcode fourni.

19.01 Préparer Xcode et la signature

Ouvrez ios/Runner.xcworkspace lorsque le workspace est présent. Sélectionnez votre équipe Apple, le bon Bundle Identifier et les profils de signature. Si le projet utilise CocoaPods et que les dépendances le demandent, lancez pod install depuis ios/. Un compte Apple Developer est nécessaire pour distribuer sur l'App Store.

19.02 Créer une archive de distribution

Générez l'archive signée puis vérifiez le résultat dans build/ios/. Testez avec TestFlight avant l'envoi en validation.

```
flutter build ipa --release
```

Attention : Le certificat, les profils et la configuration de votre compte Apple restent nécessaires ; cette commande ne garantit pas à elle seule l'acceptation sur l'App Store.

20 Publication Android, iOS et confidentialité

Préparer builds de production, signature, magasins et versionnage.

20.01 Android et Play Console

Vérifiez applicationId/package, versionCode/versionName, keystore, configuration Firebase/FCM, permissions et icônes. Compilez un AAB signé, distribuez-le d'abord en test interne, testez liens profonds, notifications et achats/paiements selon votre distribution.

```
flutter build appbundle --release
--dart-define=MUTUALBOOST_API_BASE_URL=https://mondomaine.com/api
--dart-define=MUTUALBOOST_STORAGE_URL=https://mondomaine.com/storage/
```

20.02 iOS et App Store Connect

Sur macOS, configurez Bundle Identifier, équipe Apple, signature, APNs et les capacités requises. Compilez pour iOS/TestFlight, validez la notification sur appareil physique et renseignez les métadonnées, l'assistance et la politique de confidentialité avant soumission.

```
flutter build ipa --release
--dart-define=MUTUALBOOST_API_BASE_URL=https://mondomaine.com/api
--dart-define=MUTUALBOOST_STORAGE_URL=https://mondomaine.com/storage/
```

20.03 Séparer mises à jour Laravel et binaires mobiles

La centrale signe et livre les mises à jour de code Laravel, mais pas les applications installées depuis les stores. Publiez chaque nouvelle version Flutter dans Google Play et App Store ; ne promettez pas de mise à jour native automatique par le système PHP.

20.04 Liens universels et confidentialité

Configurez assetlinks.json et apple-app-site-association avec le domaine et les identifiants d'application réels lorsqu'un lien profond est prévu. Tenez à jour les déclarations de collecte de données, permissions système et consentements liés à OneSignal, Firebase et aux services tiers.

21 Paramètres de la mutuelle et gestion quotidienne

Cotisations, produits, dépenses, rôles, adhérents et dossiers.

21.01 Identité et paramètres généraux

Après activation, modifiez coordonnées, WhatsApp, téléphone, adresse, horaires, conditions d'adhésion, délai de carence, pénalités, dates de cotisation et messages publics dans l'administration CLIENT.

21.02 Gérer les adhérents et ayants droit

Vérifiez les formulaires dynamiques d'inscription, profils, adhérents, assurés et ayants droit. Définissez les documents requis, les validations et les permissions ; testez un compte adhérent du dépôt jusqu'à l'approbation.

21.03 Trésorerie, dettes et dépenses

La gestion des cotisations, règlements, prêts, produits, demandes, budgets et dépenses dépend des rôles, des états de paiement et des règles de la mutuelle. Contrôlez les totaux avant diffusion d'un état financier. Les commandes automatisées peuvent générer des dettes ou expirer des contrats.

21.04 Documents privés et conformité

Protégez les pièces justificatives et documents dynamiques : les téléchargements sensibles doivent passer par une autorisation utilisateur, pas par un lien public. Prévoyez durées de conservation, demandes de suppression et sauvegardes chiffrées.

21.05 Offres, dons et bandeau d'état

Activez la visibilité du bouton Don si la mutuelle le souhaite. Définissez les offres et leur présentation, puis renseignez les messages par état (visiteur, à jour, non à jour, en attente) avec couleurs fond/texte ; une liste vide masque le bandeau dans Flutter.

22 Apparences, langues et contenu de vitrine

Harmoniser les cinq styles, traductions et contenus Web/mobile.

22.01 Choisir le style actif

Le Studio et les écrans d'apparence gèrent les cinq styles responsive. Choisissez la palette, les images, les composants de page et les règles d'affichage ; validez sur ordinateur, tablette et téléphone. Ne confondez pas la vitrine commerciale et la démonstration d'une mutuelle fictive.

22.02 Langues et contenus dynamiques

Le module de langues permet activation, ordre, traduction de libellés et traductions de contenus. Testez au minimum français/anglais et les autres langues activées, ainsi que l’affichage RTL lorsque l’arabe est utilisé. Un champ non traduit peut utiliser le contenu par défaut ; contrôlez visuellement ces replis.

22.03 Vitrine et SEO

Configurez les services, les avantages, les FAQ, le référencement, les images réelles et les liens Google Play/App Store. Les captures d’écran doivent correspondre aux fonctionnalités livrées ; évitez les URLs temporaires et les données personnelles réelles dans la démonstration.

22.04 Navigation et pages publiques

Validez les boutons Connexion, Inscription et Don, les liens personnalisés HTTPS, les pages légales et les écrans d’état de licence. La consultation de la vitrine reste possible si une licence doit être vérifiée, sans autoriser les écritures métier.

23 Configurer le site et l’application

Les réglages publiés depuis Laravel pilotent les interfaces prévues par la livraison.

23.01 Démarrer depuis l’un des cinq modèles

Dans Administration → Démarrer depuis un modèle, préparez l’identité, les contacts et les pages, puis enregistrez l’aperçu privé. Vérifiez-le avant de publier. La publication active le portail réel et désactive les démonstrations Web et mobile.

23.02 Choisir le bon studio

Dans Site et application, utilisez Apparence et styles pour la charte, Studio pages Web pour les pages du site, Studio pages Flutter pour les pages natives et Widgets accueil et adhérents pour leurs emplacements. Sélectionnez le style réellement publié et la bonne page avant d’enregistrer.

23.03 Configurer les langues

Activez les langues dans Langues et traductions. Traductions des contenus permet d’adapter les textes éditoriaux, les articles et les libellés métier. Testez le site et l’application dans chaque langue activée. Le nom légal de la mutuelle reste une identité propre.

23.04 Organiser les inscriptions et les droits

Dans Administration → Inscription et paiement, choisissez le format du formulaire, les étapes, le choix simple ou multiple de produits et le paiement demandé avant validation. Dans Rôles et capacités puis Affectation des rôles, attribuez les droits et contrôlez le résultat avec un compte du rôle concerné.

24 Paiements, notifications et téléchargements

Connectez les services externes à partir des réglages du backend.

24.01 Configurer les paiements

Ouvrez Cotisations et trésorerie → Passerelles de paiement et les réglages de paiement utilisés par votre circuit. Renseignez les comptes marchands, l'environnement et les URL de notification exigées par le prestataire. Personnalisez les noms, logos et l'ordre des agrégateurs. Vérifiez un paiement et sa confirmation serveur avant l'ouverture au public.

```
php artisan route:list --path=payment
```

Attention : Une page de retour réussie ne suffit pas à confirmer un encaissement. Les secrets marchands restent sur le serveur.

24.02 Configurer email, SMS et notifications Push

Dans Administration → Email, SMS et notifications, renseignez les services réellement utilisés. Vérifiez les modèles et destinataires avec un compte de test. Pour OneSignal, configurez également les capacités et paramètres propres aux projets Android et iOS de votre livraison.

24.03 Afficher les liens des stores

Dans Site et application → Téléchargements de l'application, enregistrez les liens HTTPS officiels Google Play et App Store. Chaque bouton s'affiche quand son lien valide existe. Pour le portail publié, activez aussi la section Application et téléchargements dans le projet de plateforme.

25 Mises à jour et sauvegardes

Ce parcours préserve une installation existante et ses données.

25.01 Avant toute mise à jour

Sauvegardez la base de données, les fichiers téléversés, .env et les sources de la version en service. Gardez la sauvegarde hors du dossier public. Vérifiez la restauration dans un environnement séparé et lisez la notice de la livraison.

25.02 Installer la mise à jour

Fusionnez uniquement les fichiers de la livraison dans le projet complet, en conservant .env, APP_KEY et les téléversements. Exécutez le script indiqué dans sa notice. Si composer.lock change, réinstallez les dépendances verrouillées. Appliquez des migrations uniquement lorsqu'elles sont prévues par cette mise à jour. Ne relancez pas les seeders d'initialisation.

25.03 Contrôler puis rouvrir

Videz les caches concernés et recompilez les vues, puis vérifiez connexion, droits, paiements et documents. Un worker asynchrone doit être redémarré après modification de son code. Pour Flutter, fusionnez les sources compatibles, testez et générez une nouvelle version native si le code client a changé.

```
php artisan config:clear
php artisan route:clear
php artisan view:clear
php artisan view:cache
```

25.04 Prévoir un retour arrière

Conservez les fichiers de la version précédente. Une restauration de base doit être coordonnée avec les écritures et paiements intervenus depuis la sauvegarde. Ne restaurez pas une base ancienne par-dessus des opérations récentes sans rapprochement.

26 Sécurité, journalisation et sauvegardes

Limiter les accès, protéger les secrets et valider les restaurations.

26.01 Ne pas exposer les secrets

Conservez .env, APP_KEY, clés privées de licences, clés de signatures d'updates, API keys et identifiants DB hors public/. Restreignez les permissions SSH et activez MFA chez les fournisseurs qui le permettent. Masquez les secrets lors des demandes d'assistance.

26.02 Sauvegardes 3-2-1 et restauration testée

Sauvegardez régulièrement base, pièces jointes, storage/app/private, .env et certificats/clefs. Stockez une copie hors serveur, chiffrez les copies et contrôlez l'intégrité. Exécutez au moins une restauration sur un serveur de préproduction.

26.03 Surveiller vulnérabilités et journaux

Contrôlez composer audit, les versions des packages, journaux Laravel et tâches en erreur. Une correction Composer impliquant composer.lock doit être testée hors du système automatique de ZIP applicatifs. Ne déployez pas une mise à jour sans compatibilité confirmée.

```
composer audit
php artisan about
php artisan queue:failed
php artisan migrate:status
```

26.04 Vérifier les permissions et politiques d'accès

Testez lecture/écriture pour visiteur, adhérent, gestionnaire et administrateur, notamment en licence suspendue. Les confirmations de paiement vérifiées côté serveur et l'accès administratif à la licence peuvent rester disponibles suivant la politique implémentée.

27 Dépannage et réponses aux incidents

Procédures reproductibles et sûres sans effacer les données.

27.01 Composer ne trouve pas composer.json

Cela signifie que le dossier courant n'est pas le projet Laravel complet. Recherchez composer.json, artisan et bootstrap/app.php, puis déplacez-vous dans le bon dossier. Ne créez pas un faux composer.json pour contourner l'erreur.

```
pwd
find . -maxdepth 3 -name composer.json -print
ls -l artisan composer.lock bootstrap/app.php
```

27.02 Erreur 403 : .htaccess illisible

Vérifiez la chaîne de répertoires avec namei -l, l'appartenance des fichiers, le dossier public comme racine du domaine et les journaux Apache de l'hébergeur. N'appliquez pas chmod 777 en masse.

```
namei -l /home/UTILISATEUR/mondomaine.com/public/.htaccess
ls -la public/.htaccess
```

27.03 Licence refusée / attestation invalide

Vérifiez APP_URL canonique, connexion HTTPS à la centrale, clé publique épinglée, empreinte et état d'installation avec licenses:diagnose. Ne régénérez pas les clés de signature pour tenter un dépannage.

```
php artisan licenses:diagnose --online
php artisan licenses:refresh
```

27.04 OneSignal ne livre pas les messages

Vérifiez App ID réel, secret API côté Laravel, configuration FCM/APNs, permissions, abonnement du terminal, External ID et logs HTTP. Vérifiez aussi les anciens endpoints api/v1 du code présent avant de passer en production.

27.05 Mise à jour interrompue / base en décalage

En cas d'échec, conservez l'état du journal privé, le dump SQL et les archives précédentes. Ne sortez pas de maintenance si des migrations sont en attente ; relancez les contrôles puis terminez ou restaurez explicitement.

```
php artisan migrate:status
php artisan updates:rollback --confirm=VERSION
```

27.06 Applications : affichage incohérent

Vérifiez les `--dart-define`, les caches de domaine, la version du lib/ réellement compilée, les cinq styles et le statut renvoyé par l'API. Effectuez `flutter clean` uniquement si nécessaire et ne supprimez pas `pubspec.lock` sans diagnostic.

28 Dépannage rapide

Commencez par le symptôme et vérifiez les points indiqués.

28.01 Composer introuvable ou composer.json absent

Exemple o2switch : `cd ~/mondomaine.com` puis `pwd` et `find . -maxdepth 3 -name composer.json -print`. Si aucun fichier n'apparaît, le dossier contient seulement une livraison partielle, pas le Laravel complet. Récupérez le projet d'origine : `artisan`, `composer.json`, `composer.lock`, `bootstrap/` et `public/`. Ne lancez ni `composer init`, ni `migrate:fresh`. Installer Composer en local ne résout pas l'absence des fichiers source.

```
pwd
find . -maxdepth 3 -name composer.json -print
```

28.02 Erreur 500 ou page blanche

Consultez `storage/logs/laravel.log` sur le serveur, vérifiez PHP et ses extensions, les droits d'écriture et la connexion à la base. Gardez `APP_DEBUG=false` en production. Ne publiez pas le contenu complet des journaux ou de `.env`.

28.03 Page, CSS, JavaScript ou image en 404

Vérifiez la racine `public/` du domaine, la présence du fichier et sa casse. Une image téléversée doit correspondre au stockage configuré. Conservez un `public/storage` déjà existant ; réparez le chemin concerné après sauvegarde. Les outils `VERIFIER_ASSETS_VITRINE.sh` et `VERIFIER_VUES_COMPILEES.php` sont fournis dans les livraisons qui les incluent.

28.04 Erreur de base ou seeder refusé

Vérifiez les coordonnées de `.env` et `migrate:status`. Un refus du DatabaseSeeder sur une base peuplée protège l'installation : utilisez la procédure de mise à jour. N'effacez pas les données pour contourner ce refus.

28.05 Style, widget ou bouton de store absent

Vérifiez le style publié, la page ou l'emplacement sélectionné, l'activation du bloc et ses traductions. Rechargez les réglages de l'application. Pour les stores, enregistrez une URL officielle valide et activez la section correspondante du portail.

28.06 Compilation Flutter, Gradle ou iOS en échec

Examinez flutter doctor -v et le premier message d'erreur utile. Vérifiez la compatibilité du SDK, du JDK et des fichiers Gradle fournis ; sur iOS, la signature et les dépendances natives. Ne remplacez pas des numéros de version au hasard.

28.07 L'application ne joint pas le serveur

Vérifiez baseUrl, le suffixe /api, HTTPS, le réseau de l'appareil et l'accessibilité de l'API. Une erreur 401 indique généralement un besoin de connexion ; une erreur 403 peut correspondre aux droits du compte. Ne contournez pas ces contrôles.

Références officielles

<https://laravel.com/docs/13.x>

<https://docs.flutter.dev>

<https://documentation.onesignal.com/docs/en/home>

<https://getcomposer.org/doc/>