

GUIDE OFFICIEL · CLIENT

Deploy and operate your MutualBoost platform end to end.

In-depth CLIENT documentation: prerequisites, hosting, activation, member management, Flutter, OneSignal/FCM/APNs, payments, monitoring, signed updates and recovery.

October 18, 2026 edition

28 practical chapters

Example hosts: mondomaine.com / licences.mondomaine.com

Follow this guide using the access supplied by your publisher. The license covers your installation; members then sign in with their usual accounts.

Contents

- 01. Understand both distributions
- 02. Pre-installation checklist
- 03. Before you start
- 04. Technical requirements
- 05. Install the Laravel server
- 06. Database, migrations and initialization
- 07. HTTPS hosting, storage and cron
- 08. Administrators, permissions and first login
- 09. Activate your installation license
- 10. Internal licenses, Envato and CLIENT activation
- 11. Signed software update center
- 12. Hosting and scheduled tasks
- 13. OneSignal: Android and iOS notifications
- 14. SMTP email, SMS and message templates
- 15. Payments: enablement and reconciliation
- 16. Install and connect Flutter
- 17. Flutter: complete project, API and identity
- 18. Build and publish Android
- 19. Build and publish iOS
- 20. Android/iOS publishing and privacy
- 21. Mutual settings and daily operations
- 22. Themes, languages and showcase content
- 23. Configure the website and application
- 24. Payments, notifications and downloads
- 25. Updates and backups
- 26. Security, logging and backups
- 27. Troubleshooting and incident response
- 28. Quick troubleshooting

01 Understand both distributions

Keep the publisher authority separate from each mutual organization installation.

01.01 PUBLISHER: GBAMA CODES only

The PUBLISHER distribution issues licenses, handles Envato purchases, maintains the installation portfolio and signs software releases. Install it in its own Laravel instance such as `licences.mondomaine.com`, with private keys outside the public directory.

01.02 CLIENT: each mutual organization

Each mutual organization has its own Laravel installation (`mondomaine.com`), database and configuration. Web and Flutter share that backend; the software license applies to the installation, not to each member.

01.03 Distinguish cumulative sources from full projects

The supplied ZIPs contain application sources to merge into a complete Laravel project. They omit the locked skeleton (artisan, composer.json, composer.lock); Flutter lib/ is not a standalone Android/iOS project. Do not run Composer in a directory without composer.json.

01.04 Separate environments

Use separate production, staging and development domains, databases, backups and credentials. Never connect a test app to real payment processing or copy a PUBLISHER private key into a CLIENT installation.

02 Pre-installation checklist

Prevent missing Composer files, permissions failures and incomplete deployments.

02.01 Check the project skeleton

Enter the directory containing artisan AND composer.json AND composer.lock. With only a cumulative application archive, stop and obtain the original complete Laravel project.

```
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 Validate PHP compatibility

Laravel 13 requires PHP 8.3 or a higher version supported by locked packages. Check the extensions actually used: pdo_mysql, cURL, DOM, Fileinfo, OpenSSL, Sodium, ZIP, Mbstring and XML. Composer and Artisan should run under the same PHP runtime.

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

02.03 Validate document root and permissions

Set the HTTPS domain document root to `/home/USERNAME/mondomaine.com/public`. Verify TLS, typical permissions 755 for public directories and 644 for public files; storage and bootstrap/cache must be writable by PHP.

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

02.04 Record deployment decisions

Before merging files, record the source version, CLIENT/PUBLISHER role, canonical domain, hosting provider, private backup location, and actually enabled payment/notification providers.

03 Before you start

Identify the right delivery and prepare a working environment.

03.01 Separate installation from updates

A new installation requires a complete Laravel project with artisan, composer.json, composer.lock and public/, plus a complete Flutter project with pubspec.yaml, android/ and ios/. A cumulative update or a lib-only archive must be merged into the original project; it cannot replace missing project structure.

03.02 Prepare your access

Have an HTTPS domain, hosting file and terminal access, a dedicated database and the complete source projects ready. Get the Laravel server running before connecting the mobile application. For a licensed installation, also obtain the authority URL, public key and activation key from the publisher.

03.03 Choose a test environment

Start with a separate subdomain and database. For an existing installation, go directly to Updates and backups.

04 Technical requirements

Check the server and local tools before installing dependencies.

04.01 Laravel server

Laravel 13 requires PHP 8.3 or later. `composer.lock` and `composer check-platform-reqs` determine the exact requirements of your delivery. Check `cURL`, `DOM/XML`, `Fileinfo`, `Mbstring`, `OpenSSL` and `PDO` with the appropriate database driver. The licensing module also requires PHP Sodium.

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

04.02 Flutter workstation

Install a Flutter SDK compatible with the constraints in `pubspec.yaml`, Git and Android development tools. For iOS, use a Mac with Xcode. Keep the project's `pubspec.lock` and Gradle files rather than choosing arbitrary versions.

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

05 Install the Laravel server

This workflow is exclusively for a first installation with an empty database.

05.01 1. Check complete project and install Composer

First open the original complete Laravel project, locally or on the server. Cumulative MutualBoost CLIENT/PUBLISHER archives do not include `composer.json`, `composer.lock` or `artisan`: merge them into the complete project. Check those three files BEFORE running Composer. Dependencies may be installed locally using matching PHP, but vendor built on macOS or Windows may not work on a Linux server; prefer running Composer on `o2switch` or in a compatible Linux environment. Use the same PHP runtime for Composer and 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
```

Important: If Composer says "could not find a `composer.json` file", you are in the wrong directory or have only a partial release. Do not run `composer init` or `composer update`: obtain the original complete Laravel project. Checking files locally does not create missing source files.

05.02 2. Configure the environment

Create `.env` from the supplied example only when no `.env` file already exists. Enter your domain and a new database. The following example contains placeholders, not credentials from your server.

```
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
DB_USERNAME=database_user
DB_PASSWORD="REPLACE_WITH_YOUR_PASSWORD"
```

05.03 3. Generate the key and create the tables

For this new installation only, generate APP_KEY once, check the database connection and run the migrations. Preserve this key during updates.

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

Important: On an existing server, do not regenerate APP_KEY or run commands that recreate the tables.

05.04 4. Initialize the supplied data

On a new database, the supplied DatabaseSeeder initializes settings and demonstration data. It refuses an already populated database. Run this step only if you want that initialization.

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

Important: Initialization creates fictional accounts and data. This is not an update procedure.

05.05 5. Configure the web root and files

Point the domain at the project's public/ directory. Give the PHP process write access to storage/ and bootstrap/cache/. If public/storage is absent, create the intended link. If a real directory already exists there, preserve it and its files.

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

05.06 6. Open the platform and secure the account

Open /login. In the current delivery, the seeder writes the initial credentials to storage/app/mutualboost-demo-identifiants.txt outside the public directory. Read that private file only on your server. Then update the administrator's contact details and password and configure your organization's identity.

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

06 Database, migrations and initialization

Set up MySQL without deleting an existing installation.

06.01 New database with dedicated credentials

Create a MySQL database and least-privilege account. Keep credentials in .env outside public/ and never reuse another customer's database.

```
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=musalboost_prod
DB_USERNAME=utilisateur_db
DB_PASSWORD="REPLACER_SECRET"
```

06.02 Composer and initial bootstrap

Install dependencies from composer.lock and validate Laravel startup. If package:discover fails, inspect bootstrap/app.php and bootstrap/cache instead of immediately running composer update.

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

06.03 Consolidated migrations and history

On an empty database, run migrate --force and inspect migrate:status. On an existing database, back up first: never use migrate:fresh, db:wipe or manually edit the migrations table. Historical migrations may have been archived after consolidation.

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

06.04 Seeders: use the correct workflow

DatabaseSeeder initializes pages, settings and demo content on a new installation. It may reject populated databases; do not force it in production. DemoMutualSeeder expects setup by DatabaseSeeder and may require Faker to be installed.

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

Important: Do not run DemoMutualSeeder by itself and never publish demo accounts on a real mutual organization.

06.05 APP_KEY encryption key

Generate APP_KEY once for a brand-new installation. It protects encrypted data; regenerating it on a live installation can make existing encrypted records unreadable.

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

07 HTTPS hosting, storage and cron

Document root, files, scheduling and monitoring.

07.01 Public directory and storage link

Serve public/index.php. If public/storage does not exist, use php artisan storage:link; if a real directory already contains files there, do not delete it without checking and backing up.

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

07.02 Laravel cron: one entry per installation

Create ONE cron job in cPanel or crontab invoking schedule:run every minute with the installation PHP. schedule:list lists planned tasks but does not prove that system cron is installed.

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

07.03 Review scheduled jobs

Check contribution reminders, debt generation, payment reconciliation, contract expiry, automated notifications, license refresh and daily release checks. Read the exact schedule from schedule:list on the installed server.

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

07.04 Queues and mail: environment-dependent

If QUEUE_CONNECTION is database/redis, provide a supervised worker or a hosting-compatible execution method. If using sync, no async worker runs; account for notification and response-time impact.

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

Important: Run persistent queue workers only if your host and configuration support them.

07.05 HTTPS, backups and logs

Enable TLS, HTTP→HTTPS redirects, APP_DEBUG=false, private logs and database+file backups. Periodically restore on staging. Never expose .env, storage/app/private, logs or SQL dumps.

08 Administrators, permissions and first login

Initial accounts, security, roles and public journeys.

08.01 Provision an authorized administrator

The /login page must remain usable before license activation so the real administrator can reach /admin/license. DefaultAdminSeeder can create an initial account on demo installations only; rotate its password and disable test access before production.

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

Important: Never publish default login credentials on a public documentation page.

08.02 Roles and capabilities

Create named roles (secretary, treasurer, manager) and grant only required capabilities. Reserve licensing, payment credentials, updates and role management for authorized personnel.

08.03 Smart public navigation

Under Administration → Website and app → Login/Registration/Donation buttons, configure public actions. The navigation adapts to Guest, Member or Administrator; registration also respects the enrollment policy and license restrictions.

08.04 Disable demo accounts

Keep fictional mutual accounts limited to isolated environments. Verify permissions, absence of real financial actions in demo mode, and secure sign-out via POST/CSRF.

09 Activate your installation license

Your mutual organization uses the client role. The publisher supplies the authority connection details.

09.01 1. Receive the correct project and license information

Your installation combines the complete Laravel project, compatible cumulative updates and, if you use the mobile app, the complete Flutter project. Obtain the authority's final HTTPS URL, public key and your internal license key or CodeCanyon purchase code from the publisher. A cumulative archive does not replace missing project structure.

Important: Client means your organization's platform. Its members then use your website and app; each member does not activate a separate software license.

09.02 2. Configure the client role only

In your organization's Laravel .env file, set your domain and the public connection details supplied by the publisher. The example does not identify an existing authority. Clear the configuration cache after editing.

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

Important: After saving .env, run `php artisan config:clear` in the Laravel project terminal. The Envato author token and private signing key stay with the publisher; neither is required on your server or in Flutter.

09.03 3. Enter the key in administration

Sign in at `/admin/license`: Administration → Platform license. Choose Internal MutualBoost license for a key issued by the publisher, or CodeCanyon / Envato for a purchase code. Enter it, select Activate license and check the domain and Active license status. Member accounts and ADMIN-DEMO cannot activate a license.

Important: The purchase code is available with the license certificate in your Envato downloads. Your Envato account password is not requested.

09.04 4. Check renewal and service continuity

Keep the Laravel cron configured on your hosting. The server verifies the signed certificate locally during navigation and attempts renewal when it reaches 24 hours. A temporary outage leaves an unexpired certificate usable for up to seven days, within any internal license expiry. Use Verify now after resolving the issue.

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

Important: The end of Envato support is shown separately from usage rights. This module does not convert the support date into a license expiry.

09.05 5. Transfer the platform to another domain

Before moving, open Transfer or replace license at `/admin/license`, confirm deactivation, then activate the new server with its canonical domain. Keep the private identity for a normal update on the same domain. Ask the publisher to release the installation if the old server is unavailable.

Important: Do not clone an activated installation to operate another platform. Capacity depends on the license; a CodeCanyon purchase is managed with one active installation per end product.

09.06 6. Understand a restriction and contact support

An outdated certificate, an internal expiry or a suspension can pause new website and API writes with HTTP 423. Viewing, account recovery, administrator license access and callbacks for previously initiated payments remain available. Give the publisher the domain, displayed message and issue time, without publishing your full key, .env or passwords.

Important: Reading this guide does not configure a real license authority or activate an Envato purchase. Obtain connection details from your publisher.

10 Internal licenses, Envato and CLIENT activation

Understand duration, attestations and controls without blocking activation.

10.01 Activate directly in CLIENT Laravel

At /admin/license, an authorized administrator enters an internal license key or Envato purchase code. The server contacts the HTTPS authority, pins its public key on first trusted use and validates the signed certificate. Never enter the signing private key into CLIENT settings.

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

10.02 Understand license durations

An internal license without an end date is perpetual; Envato use rights are distinct from support. The technical certificate is time-limited (up to seven days) and refreshed periodically: the certificate refresh date is not the commercial license expiry.

10.03 Diagnose without disclosing secrets

Check the APP_URL domain, published fingerprint, pinned public key, certificate and installation identity. A fingerprint is not the complete public key. Never delete client.json or authority.json to force activation.

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

10.04 Suspension and migration

Suspension/revocation is managed at the authority and does not erase historical operations. For an actual domain change, prepare backups and use the installation release/transfer flow rather than silently copying a certificate to a different server.

11 Signed software update center

Check, back up, deploy and restore without browser-driven code execution.

11.01 Bootstrap the updater once

Bootstrap the updater by trusted file transfer, not by an updater that is not yet installed. Check Sodium and ZIP. The updates public key differs from the license key and must be pinned as MUTUALBOOST_UPDATES_PUBLIC_KEY on CLIENT.

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

11.02 Check before download

The /admin/updates screen shows version and updates. updates:check compares signed manifests. An active CLIENT can download through updates:download; Ed25519 signature, target distribution and SHA-256 are checked before copying.

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

11.03 Deploy from SSH with a backup

File deployment is started only with version-specific SSH confirmation. Back up DB and files. For releases with migrations, provide a SQL backup path, run migrate --force, check records and finalize with 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 Recovery and rollback limits

If deployed code fails, use updates:rollback --confirm=VERSION and validate service. Source rollback does NOT restore changed tables or Composer dependencies. Restore a tested database backup separately when SQL must be reverted.

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

11.05 Scheduled checks, never unattended code installs

updates:check runs daily from routes/console.php. Cron must not call updates:apply or accept arbitrary browser-uploaded code ZIPs. Keep a release log with version, publisher, notes, hashes and restoration status.

12 Hosting and scheduled tasks

Prepare the domain, scheduler and background processing when needed.

12.01 cPanel and o2switch

Select compatible PHP versions for both the domain and the terminal. Set the document root to public/, enable HTTPS and create the database and its user with appropriate access. Replace php with the full PHP executable path when your hosting requires it.

12.02 Laravel scheduler

Add one cron entry running every minute and replace both paths in the example. Use `schedule:list` to check that the delivery's tasks are actually registered. These include payment reconciliation, contract expiry and reminders.

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

Important: Check with `php artisan schedule:list`. The project's timezone determines scheduled execution times.

12.03 Background queues

If your configuration uses an asynchronous queue, arrange a supervised queue:work process managed by your hosting provider. A synchronous configuration does not require that process. Check the delivery's queue configuration before enabling it.

13 OneSignal: Android and iOS notifications

OneSignal app, FCM, APNs, Laravel-side secrets and Flutter delivery.

13.01 Create a OneSignal application

Create a OneSignal app for the organization. Under Settings → Keys & IDs, obtain the App ID (UUID, safe to use in SDK setup) and REST API Key (SECRET, server-side only). Use distinct staging and production credentials.

13.02 Configure Android and FCM

In Firebase Console, configure the Android project with the exact app package ID, enable FCM and link the required service credentials in OneSignal. Add `google-services.json` only according to the project's native integration; never embed a server secret inside Flutter.

13.03 Configure Apple Push Notification service

In Apple Developer, use the correct Bundle ID, enable Push Notifications and provide OneSignal with an APNs key or certificate according to its instructions. Enable the required Xcode Push Notifications and Background Modes/Remote notifications capabilities and test on a physical iPhone.

13.04 Configure Laravel and protect the REST API Key

In Administration → General settings → API / Notifications, set `onesignal_app_id` and `onesignal_rest_api_key` and enable `onesignal_enabled` only after validation. Laravel rejects placeholder credentials; an empty secret field must not overwrite the saved key. Never expose the REST key in public HTML or Flutter APIs.

13.05 Initialize the Flutter SDK and user identity

Keep the `onesignal_flutter` dependency from `pubspec.yaml`, initialize the SDK with the non-secret App ID, request permission contextually and link the signed-in user to the external ID expected by Laravel. Disassociate the identity on sign-out according to the installed SDK API.

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

13.06 Test each notification route

Test on two devices: permission, subscription token, direct notification, targeted user, contribution reminder, deep link and sign-out. The supplied Laravel controllers still include legacy OneSignal calls (`api/v1 + include_external_user_ids`); validate compatibility with the current OneSignal API before claiming push is production-ready.

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

13.07 Respect preferences and privacy

Publish a privacy notice and obtain the relevant consents. Do not put medical diagnoses or sensitive financial data in lock-screen notification text. Validate app-store privacy declarations.

14 SMTP email, SMS and message templates

Configure actual providers, sender identities and business events.

14.01 Email: working SMTP

In General settings → API, set `smtp_host`, `smtp_port`, `smtp_username`, `smtp_password`, `smtp_encryption`, `smtp_from_address` and `smtp_from_name`. Hosting must allow outbound connectivity. Configure SPF/DKIM/DMARC with the mail provider and send a test message.

14.02 SMS: select a provider

The provider selector includes Twilio, Infobip, Orange SMS API or a custom gateway. Set `sms_provider`, `sms_api_key`, `sms_api_secret` and `sms_sender_id` according to the provider contract; test authorized delivery before enabling bulk reminders.

14.03 Customize business notification events

Per event (`welcome`, `payment`, `approval`, `rejection`, `new_dependent`, `death_notice`, `reminder`), enable `notify_*_email`, `notify_*_sms` and `notify_*_push`. Customize subject and message bodies and test a real event with a test account.

14.04 Monitor failures

Inspect Laravel logs, failed queue jobs, provider quotas, authentication failures and HTTP errors. Never copy REST API keys into shared diagnostic reports.

15 Payments: enablement and reconciliation

Configure payment processors without confusing branding with API authentication.

15.01 Select available integrations

The code includes settings for Wave, CinetPay, TouchPay, PayPal and Flutterwave; some workflows also include FedaPay, Jeko and Stripe settings. Enable only payment services actually integrated, contracted and tested.

15.02 Configure server-side credentials

Under General settings → Payments, enter provider credentials: Wave API key/signing secret; CinetPay Site ID/API key/secret; TouchPay agency/authentication; PayPal Client ID/secret/environment; Flutterwave secret. Public logos, labels and order do not replace technical credentials.

15.03 Returns, webhooks and duplicate payments

Configure return, cancel and webhook URLs according to the actual routes and provider dashboard. Server-side verify signature/reference, amount, currency, status and idempotency; never treat a browser redirect as proof of payment.

15.04 Run full sandbox tests

Test approved, failed, canceled and delayed payments, repeated callbacks, network loss and payments:reconcile-pending reconciliation. Review logs without exposing card details, secrets or health records.

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

16 Install and connect Flutter

Start when the HTTPS server and its APIs are available.

16.01 1. Open the complete project

Open the directory containing pubspec.yaml. Merge lib updates into that project. Install dependencies and review diagnostics before building.

```
flutter pub get
flutter analyze
```

16.02 2. Enter your organization's server

In the verified Flutter source, `lib/services/api_service.dart` defines `baseUrl` and `storageUrl`. Replace both addresses with your server URLs. Keep `/api` for the API and `/storage/` for public files. Another version may centralize these values elsewhere; locate their definitions.

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

Important: Flutter connects to your client platform. Do not replace `baseUrl` with the license authority URL.

16.03 3. Configure the application identity

Check the display name, icons, Android application ID and iOS Bundle Identifier in the supplied native projects. Use your own identifiers for a new app. Keep the identifiers of an app that is already published.

16.04 4. Test on a device

Detect your device or emulator and launch the application. Check the home page, authentication, media, languages and contribution history using test data.

```
flutter devices
flutter run
```

Important: The device must be able to reach your HTTPS domain. `localhost` on a phone refers to that phone, not your computer.

17 Flutter: complete project, API and identity

Connect Dart sources to Laravel while preserving style variants.

17.01 Get the complete mobile project

Start from the complete Flutter repository containing `pubspec.yaml`, `pubspec.lock`, `lib/`, `android/`, `ios/` and `assets`. A cumulative `lib/` ZIP cannot build binaries alone; back up and merge `lib/` into the original project.

17.02 Configure your backend URLs

Use the compilation defines supported by this MutualBoost build; never keep a hard-coded demo URL. API and media URLs must point to the same HTTPS CLIENT instance.

```
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 Shared login, roles and licensing

Authentication tokens, roles and permissions come from Laravel. Activation at `/admin/license` enables both site and app according to access rights. Flutter must display license states without bundling private keys or bypassing Laravel verification.

17.04 Five styles and status messages

Preserve five design models and dynamic colors. The status-message ticker appears directly below each AppBar, scrolls horizontally, treats each line as one message and vanishes if none is provided. Background/text colors depend on the member state and come from Laravel.

17.05 Test end-to-end integration

On Android and iPhone, test login, allowed registration, member screens, roles, inactive/active license, notifications, media, sandbox payments, translations and style switching. Run flutter analyze after each merge.

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

18 Build and publish Android

Prepare production signing before distributing the application.

18.01 Signing and configuration

Configure your signing key in the Android project using the official Flutter procedure. Keep a private backup of the key and its passwords. Also check the application version, permissions and notification configuration.

18.02 Generate distribution files

Use the AAB for Google Play publication and the APK for direct distribution testing. Typical output directories are `build/app/outputs/bundle/release/` and `build/app/outputs/flutter-apk/`. Test the release build before publishing.

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

Important: Building does not automatically publish the app. Upload the AAB to your own Google Play Console account and complete the store listing.

19 Build and publish iOS

Prepare iOS distribution on macOS using the supplied Xcode project.

19.01 Prepare Xcode and signing

Open ios/Runner.xcworkspace when the workspace is available. Select your Apple team, the correct Bundle Identifier and signing profiles. If the project uses CocoaPods and dependencies require it, run pod install from ios/. An Apple Developer account is required for App Store distribution.

19.02 Create a distribution archive

Generate the signed archive and check the build/ios/ output. Test through TestFlight before submitting for review.

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

Important: Your certificate, profiles and Apple account configuration are still required. This command alone does not guarantee App Store approval.

20 Android/iOS publishing and privacy

Prepare release builds, signing, stores and versioning.

20.01 Android and Play Console

Verify applicationId/package, versionCode/versionName, keystore, Firebase/FCM configuration, permissions and app icons. Build a signed AAB, distribute in internal testing first, and validate deep links, notifications and any purchases/payments.

```
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 and App Store Connect

On macOS, set Bundle Identifier, Apple team, signing, APNs and required capabilities. Build for iOS/TestFlight, validate notifications on a real device, and complete metadata, support and privacy policy before submission.

```
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 Separate Laravel updates and mobile binaries

The authority signs and delivers Laravel code updates, not binaries distributed through stores. Publish every Flutter binary version through Google Play/App Store; the PHP updater does not remotely patch native apps.

20.04 Universal links and privacy

Configure `assetlinks.json` and `apple-app-site-association` with your real domain and app identity if deep links are supported. Update platform privacy declarations, permissions and consents related to OneSignal, Firebase and third-party services.

21 Mutual settings and daily operations

Contributions, products, expenses, roles, members and cases.

21.01 Organization identity and settings

After activation, configure contact details, WhatsApp, phone, address, hours, enrollment rules, waiting periods, penalties, contribution dates and public messages in CLIENT administration.

21.02 Manage members and dependents

Review dynamic registration, profile, member, insured-person and dependent forms. Configure required documents, approvals and permissions; test a member account from submission through approval.

21.03 Treasury, debt and expenses

Contributions, payments, loans, products, requests, budgets and expenses depend on roles, payment status and organization rules. Reconcile totals before publishing reports. Scheduled commands may generate debts or expire contracts.

21.04 Private documents and compliance

Protect supporting documents and dynamic document exports: sensitive downloads must enforce authorization, not rely on public URLs. Define retention periods, deletion requests and encrypted backups.

21.05 Offers, donations and status banner

Enable the Donation button only if the organization wants it. Configure offers and their presentation, then set status messages for guest, paid-up, overdue and pending users with text/background colors; an empty message list hides the ticker in Flutter.

22 Themes, languages and showcase content

Keep five styles, translations and Web/mobile content consistent.

22.01 Select an active style

The Studio and appearance screens support five responsive themes. Configure palette, imagery, components and display rules; validate desktop, tablet and mobile. Keep the commercial showcase distinct from the fictional mutual demo.

22.02 Languages and dynamic content

The language module supports activation, ordering, UI strings and content translations. Test at least French/English and every enabled language, plus RTL layouts when Arabic is used. Untranslated content may fall back to defaults; verify those fallbacks visually.

22.03 Public showcase and SEO

Configure services, benefits, FAQs, SEO, actual screenshots and Google Play/App Store links. Screenshots should reflect shipped capabilities; avoid temporary URLs and real personal data in the demo.

22.04 Navigation and public pages

Verify Login, Registration and Donation buttons, custom HTTPS links, legal pages and license state screens. The public showcase may remain readable while license verification is required, without allowing restricted business writes.

23 Configure the website and application

Settings published from Laravel drive the interfaces supported by your delivery.

23.01 Start from one of the five templates

Under Administration → Start from a template, prepare the identity, contacts and pages, then save the private preview. Review it before publishing. Publication activates the live portal and disables public Web and mobile demonstrations.

23.02 Choose the right studio

Under Website and application, use Appearance and styles for the visual identity, Web page studio for website pages, Flutter page studio for native pages and Home and member widgets for supported placements. Select the published style and the correct page before saving.

23.03 Configure languages

Enable languages in Languages and translations. Content translations lets you adapt editorial text, articles and business labels. Check the website and application in every enabled language. The organization's legal name remains its own identity.

23.04 Organize registration and permissions

Under Administration → Registration and payment, choose the form layout, steps, single or multiple products and payment requirements before approval. Use Roles and capabilities and Assign roles to grant access, then check the result with an account holding that role.

24 Payments, notifications and downloads

Connect external services through the backend settings.

24.01 Configure payments

Open Contributions and treasury → Payment gateways and the payment settings used by your workflow. Enter merchant accounts, environment and provider notification URLs. Customize gateway names, logos and ordering. Verify a payment and its server confirmation before launch.

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

Important: A successful return page alone does not confirm receipt of funds. Merchant secrets stay on the server.

24.02 Configure email, SMS and Push notifications

Under Administration → Email, SMS and notifications, configure the services actually used. Check templates and recipients with a test account. For OneSignal, also configure the native Android and iOS capabilities and settings required by your delivery.

24.03 Display store links

Under Website and application → Application downloads, save the official HTTPS Google Play and App Store links. Each button appears when its valid link is present. For the published portal, also enable the Application and downloads section in the platform project.

25 Updates and backups

This workflow preserves an existing installation and its data.

25.01 Before every update

Back up the database, uploaded files, .env and the deployed source version. Store backups outside the public directory. Check restoration in a separate environment and read the delivery instructions.

25.02 Install the update

Merge the delivery files into the complete project while keeping .env, APP_KEY and uploads. Run the installer specified in its instructions. If composer.lock changes, reinstall locked dependencies. Apply migrations only when the update requires them. Do not rerun initialization seeders.

25.03 Check and reopen

Clear the relevant caches, compile views and verify authentication, permissions, payments and documents. Restart asynchronous workers after their code changes. For Flutter, merge compatible sources, test and build a new native release when client code changes.

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

25.04 Plan rollback

Keep the previous version's files. Coordinate database restoration with transactions and payments recorded since the backup. Do not overwrite recent operations with an older database without reconciliation.

26 Security, logging and backups

Limit access, protect secrets and validate restores.

26.01 Protect credentials

Keep .env, APP_KEY, license signing private keys, update signing keys, API keys and DB credentials outside public/. Restrict SSH permissions and enable provider MFA when available. Redact secrets in support requests.

26.02 3-2-1 backups and restore tests

Back up the DB, uploads, storage/app/private, .env and certificates/keys regularly. Keep an off-server encrypted copy and check integrity. Perform at least one restore to staging.

26.03 Audit dependencies and logs

Check composer audit, package versions, Laravel logs and failed jobs. Composer upgrades changing composer.lock must be tested outside the application ZIP updater. Do not deploy unvalidated dependencies.

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

26.04 Verify permissions and access policies

Test read/write access for guests, members, managers and administrators, including suspended-license states. Server-verified payment confirmations and administrator access to licensing may remain available according to the implemented policy.

27 Troubleshooting and incident response

Reproducible and safe diagnostics without wiping data.

27.01 Composer cannot find composer.json

The current directory is not the complete Laravel project. Locate composer.json, artisan and bootstrap/app.php, then cd to the correct folder. Do not create a fake composer.json to bypass the error.

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

27.02 403: unreadable .htaccess

Use namei -l to inspect directory traversal, ownership, the domain's public document root and Apache host logs. Never apply chmod 777 recursively.

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

27.03 License refused / invalid certificate

Check canonical APP_URL, HTTPS authority reachability, pinned public key, fingerprint and installation status using licenses:diagnose. Never rotate signing keys as a quick fix.

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

27.04 OneSignal is not delivering

Check real App ID, Laravel API secret, FCM/APNs settings, device permission, subscription, External ID and HTTP logs. Validate the legacy api/v1 calls in this code before going live.

27.05 Interrupted update / schema mismatch

On failure, retain private update journal, SQL dump and prior sources. Do not exit maintenance if migrations remain pending; validate and then explicitly complete or restore.

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

27.06 App: inconsistent screens

Check dart-defines, domain-scoped caches, actual compiled lib/ version, five styles and API-reported status. Use flutter clean only when necessary; do not delete pubspec.lock without a diagnosis.

28 Quick troubleshooting

Start with the symptom and check the relevant points.

28.01 Composer manifest missing (composer.json)

On o2switch, run `cd ~/mondomaine.com`, `pwd`, then `find . -maxdepth 3 -name composer.json -print`. If none appears, you have an incomplete release rather than a complete Laravel project. Restore the original artisan, `composer.json`, `composer.lock`, `bootstrap/` and `public/` files. Never run `composer init` or `migrate:fresh` to bypass this; installing Composer locally cannot recreate missing source files.

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

28.02 HTTP 500 or a blank page

Inspect `storage/logs/laravel.log` on the server. Check PHP, extensions, directory write access and database connectivity. Keep `APP_DEBUG=false` in production. Do not publish complete logs or `.env` contents.

28.03 A page, stylesheet, script or image returns 404

Check the domain's `public/` document root, file presence and filename case. Uploaded images must match the configured storage. Preserve an existing `public/storage` directory and repair the affected path after backup. Deliveries may include `VERIFIER_ASSETS_VITRINE.sh` and `VERIFIER_VUES_COMPILEES.php` for diagnostics.

28.04 Database errors or a refused seeder

Check `.env` database settings and `migrate:status`. DatabaseSeeder refusing a populated database protects the installation; use the update workflow. Do not delete data to bypass that refusal.

28.05 A style, widget or store button is missing

Check the published style, page or placement, block activation and translations. Reload the application settings. For store buttons, save a valid official URL and enable the matching portal section.

28.06 Flutter, Gradle or iOS build errors

Inspect `flutter doctor -v` and the first relevant error. Check SDK and JDK compatibility with the supplied Gradle files; on iOS, check signing and native dependencies. Do not replace version numbers arbitrarily.

28.07 The app cannot reach the server

Check `baseUrl`, the `/api` suffix, HTTPS, device connectivity and API availability. HTTP 401 generally indicates authentication is needed; HTTP 403 may reflect account permissions. Do not bypass these controls.

Official references

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

<https://docs.flutter.dev>

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

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