

**Composant Serveur API REST pour WinDev®**

Léger.

Rapide.

Gratuit & Open Source.

Sans serveur IIS/Apache/...

Sans licence Serveur WebDev®.

Pas de SAAS, pas d'abonnement.

Windows 32/64 & Linux

Présentation

NEW : LightREST V3

La Documentation



Téléchargements

Un coup de main ?

LightREST V3 marque une évolution majeure : **plus ouvert, plus flexible, plus sécurisé, et plus robuste en production.**

Cette page récapitule les nouveautés principales et surtout **ce qu'elles apportent concrètement** au quotidien.

LightREST devient open source

Le code intégral du composant **WinDev®** est livré, ainsi que celui du moteur REST développé en **GO** (choisi pour ses performances). Pour être utile à un maximum de WinDeveloppeurs, le projet LightREST est livré en WinDev® 25, évidemment migrable dans les versions ultérieures. Le projet GO est compilé en version 1.25.5.

Le code source de LightREST est diffusé sous licence **MIT**,

donc sans aucune restriction de modification ni aucune obligation de diffusion. Evidemment, toutes les suggestions de corrections, évolutions, améliorations proposées par la communauté des développeurs seront les bienvenues !

- **Confiance & pérennité** : plus de "boîte noire". Le développeur LightREST peut auditer, comprendre, et maintenir.
- **Débogage accéléré** : en cas de cas limite, suivi du flux complet (Wlangage → moteur Go → réponse).
- **Évolutivité** : Adaptation du composant aux contraintes spécifiques (infra, sécurité, conventions internes).
- **Performances maîtrisées** : Go est particulièrement efficace pour les serveurs réseau (I/O, concurrence, throughput).

Multi-instances

Possibilité d'instancier **plusieurs serveurs** LightREST dans un même exécutable (sur des ports différents) :

- **Séparation des usages** : un port "public", un port "admin", un port "interne".
- **Politiques différentes** : timeouts, limites, sécurité, logs... selon le serveur.
- **Déploiement simplifié** : un seul binaire, plusieurs endpoints (par exemple public/privé/admin), sans multiplier les services.

Hooks before / after

Les hooks sont de puissants mécanismes d'interception d'événements clés : réception, authentification, avant

exécution, après exécution, avant envoi.

Un hook peut **refuser** une requête, **compléter ou remplacer** une réponse (corps, entêtes, ...) et :

- **Centralise** les traitements récurrents comme l'auth, l'audit, le rate-limit, les headers de sécurité.
- **Évite la duplication** : plus besoin de répéter les mêmes contrôles dans chaque route.
- **Standardise les réponses** : enveloppe JSON, format d'erreur homogène, IDs de corrélation, etc.
- **Facilite l'observabilité** : logs, métriques, traces, sans polluer les handlers métier.

Plus de détails sur le fonctionnement des Hooks ICI:

Bon réflexe : utiliser les hooks comme "pipeline transverse", et garder les routes centrées sur le métier.

ID Cipherring

Ajout d'un mécanisme de **chiffrement** des IDs (ex: IDs BDD) pour éviter qu'ils soient prévisibles/exploitable.

- **Réduit le risque d'énumération** (ex: tester /users/1, /users/2, /users/3...).
- **Protège la valeur métier** des identifiants internes (clients, devis, factures, etc.).
- **Bloque** le "data scraping" automatisé.

Important : l'ID Cipherring est une **défense en profondeur**. Il ne remplace pas l'auth/ACL : il la complète.

Optimisations

Améliorations internes du moteur LightREST : variables

atomiques (thread safety), maps concurrentes, mutex, channels synchronisés, time-bounds. Traitements Interne du moteur jusqu'à 50% plus rapides.

- **Plus de débit** pour le même serveur (ou moins de ressources pour le même débit).
 - Gestion des réponses au **format binaire** jusqu'à 50% plus rapide
 - **Meilleure tenue en charge** lors des pics.
 - **Meilleure latence** et UX côté clients (mobile, web, intégrations).
-

Sécurisation renforcée

Renforcement de l'isolation entre traitements parallèles (compartimentation des exécutions).

- **Réduit les effets de bord** entre requêtes concurrentes (données partagées, contamination de contexte).
 - **Diminue les risques de race conditions** et bugs "fantômes" difficiles à reproduire.
 - **Améliore la stabilité** sous charge, là où la concurrence révèle les faiblesses.
-

Windows 32 bits

Le composant fonctionne désormais en **Windows 32 bits**.

- **Compatibilité** avec des environnements legacy ou des contraintes d'exécution spécifiques.
 - Possibilité d'embarquer LightREST dans des binaires 32 bits existants sans refonte majeure.
-

Multi OS

Version disponibles :

- **Complète** (multi OS)
- **Windows**
 - **32 bits**
 - **64 bits**
 - **32 bits + 64 bits**
- **Linux 64 bits**

Pourquoi utiliser la version du composant correspondant à l'OS cible ?

- **Optimisation footprint** : binaire plus petit, moins de mémoire occupée.
 - **Packaging plus propre** selon les cibles (prod Linux, dev Windows...).
 - **Déploiement plus rapide** (transferts, CI/CD, containers).
-

MaxRequests

Définition d'un maximum de requêtes simultanées en exécution pour éviter l'overflow.

- Protection directe contre la **surcharge** (pics, bots, erreurs clients).
 - Évite l'épuisement des ressources (threads/mémoire) et l'effet domino.
 - Permet un comportement contrôlé (ex: refus propre, réponse cohérente).
-

CustomData serveur & route

Des données propres au serveur et/ou à la route sont transmises automatiquement à la requête (lrRequest). CustomData est implémenté dans lrServer et lrRoute et transmis automatiquement aux handlers REST avec chaque requête. Evite par exemple de recharger à chaque requête des éléments globaux, des paramètres, ..., et ainsi d'optimiser le temps d'exécution.

- **Éviter de relire** en boucle des infos globales (config, références, caches, services).
- **Injecter du contexte** propre à une route : permissions requises, tags, options, stratégie de réponse...
- **Architecture plus propre** : le runtime porte le contexte, les handlers restent "métier".

Bon réflexe : y stocker des éléments susceptibles de causer des collisions entre des requêtes concurrentes (pas besoin de sémaphores ou sections critiques)

WinDevErrorDetails

Contrôle du **niveau d'information** renvoyé au client en cas d'erreur.

- En production, pas de **leak** des détails techniques (stack, SQL, chemins, lignes de code, variables, infos internes).
- Erreurs **propres côté client** (messages maîtrisés, codes cohérents).
- Ajustement du niveau de détail selon l'environnement : **dev** (complet) vs **prod** (minimal).

Request timeout (global ou par route)

Définition un timeout : – pour tout le serveur, – et/ou spécifique

à une route (traitement long).

- Évite les traitements qui tournent **indéfiniment** (protection prod).
 - **Préserve les ressources** serveur (threads, I/O, mémoire).
 - Permet d'adapter : routes rapides (timeout court), exports lourds (timeout long).
-

Taille du composant

Le composant V3 est plus compact qu'en V2, d'environ 40% :

- Déploiements plus rapides (CI/CD, transferts, instanciations).
 - Moins d'empreinte sur les environnements contraints.
 - Meilleure "densité" si on multiplie les instances / services.
-

Read / Write timeout

Détection de la fermeture de connexion HTTP côté client lors de la lecture/écriture. Si le client est parti, le traitement est **annulé automatiquement**.

- Évite une **charge inutile** (on ne génère pas une réponse que personne ne recevra).
 - Protège contre les clients instables / réseaux mobiles.
 - Améliore la résilience face à certains comportements "lents" (idle, connexions rompues).
-

Idle State

- Permet de garder la connexion HTTP ouverte entre le client et le serveur LightREST après exécution d'une requête

- La connexion suivante évite la réouverture de la connexion (et la renégociation SSL en mode HTTPS)
 - La connexion est refermée automatiquement lorsque le IDLE Timeout est atteint (60s par défaut)
 - Résultat : Gain de performance lors de requêtes successives
-

Objet lrRoute

Le nouvel objet lrRoute permet de décrire les routes de façon plus riche.

- Centralise les métadonnées d'une route (options, custom data, politiques...).
 - Facilite la génération de documentation, tests, et outillage (ex: introspection).
 - Permet d'attacher des hooks spécifiques à la route
-

Conclusion

Avec la V3, LightREST devient :

- **plus transparent** (open-source),
 - **plus industrialisable** (hooks, multi-instances, policies),
 - **plus sûr** (cipher ID, erreurs maîtrisées, compartimentation),
 - **plus robuste** en prod (timeouts, maxrequests, déconnexions),
 - et **plus léger** (taille réduite, variantes OS).
-

© CODE LINE 2018-2026

