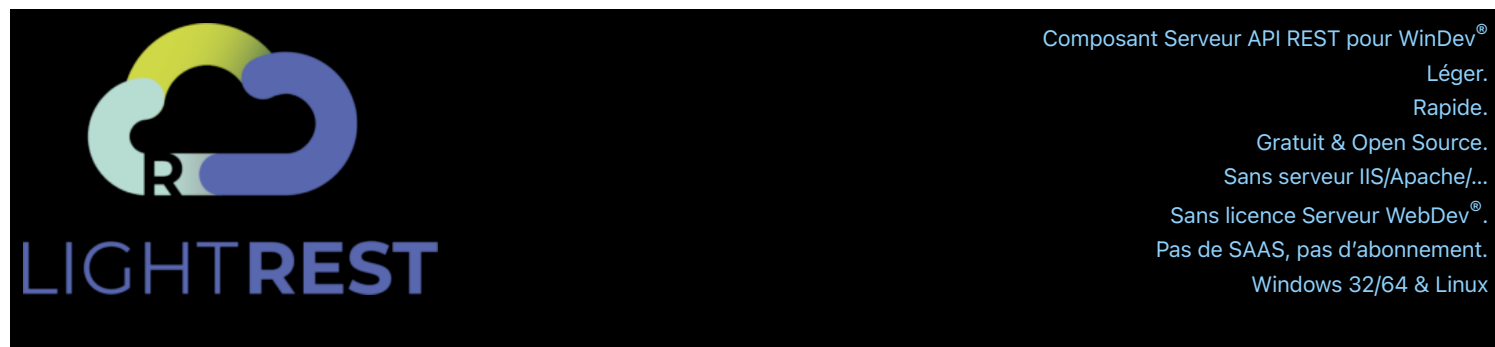
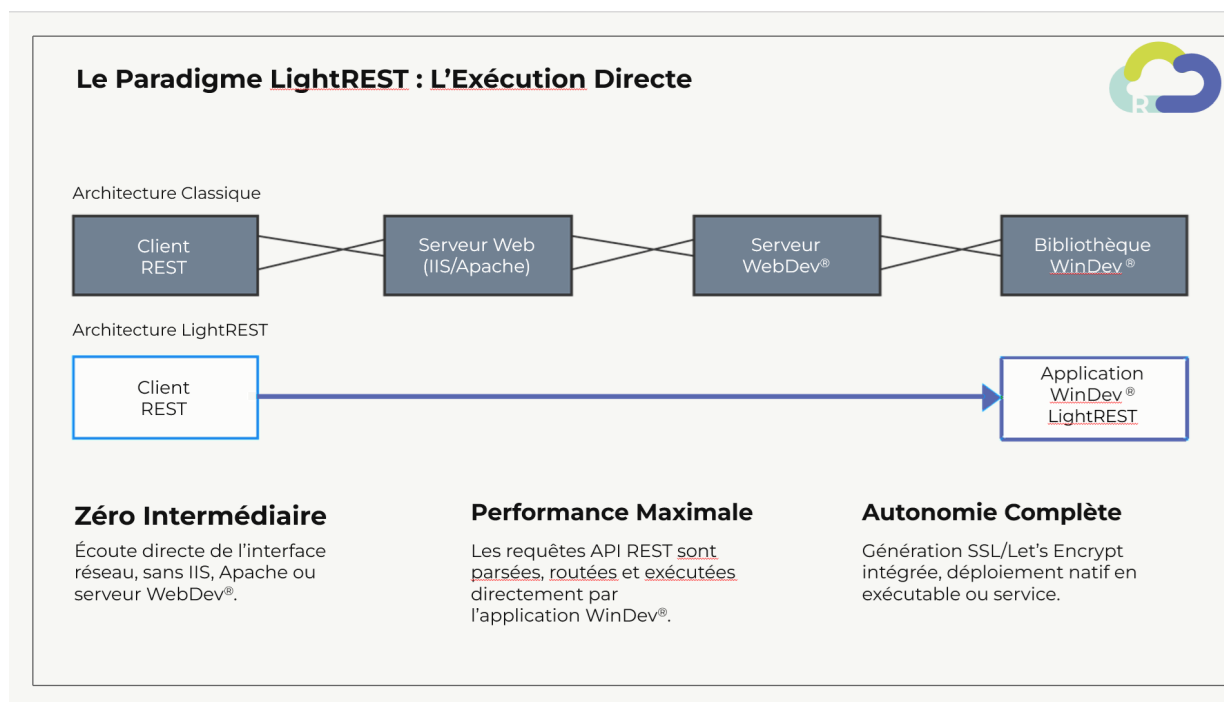




LightREST est un composant gratuit et open source est destiné aux développeurs WinDev® souhaitant créer un Web Service REST (**RE**presentational **ST**ate **TR**ansfer).

Contrairement au système proposé nativement par WinDev®, **LightREST** ne nécessite **ni l'installation d'un serveur Web** (IIS, Apache, ...) **ni l'acquisition et les mises à jour d'un serveur WebDev®**. Les requêtes API REST sont exécutées directement par votre application WinDev® sans subir un traitement chronophage par le serveur Web puis le serveur WebDev®. Un serveur **LightREST** écoute lui-même l'interface réseau et répond directement aux requêtes REST, sans aucun intermédiaire.



Disponible sous la forme d'un composant Windows 32 et 64 bits et Linux 64 bits, compatible avec toutes les versions WinDev® à partir de la V25, **LightREST** a été développé et utilisé depuis plusieurs années par la société CODE LINE, qui se fait un plaisir de le mettre à la disposition de la communauté des développeurs. La version 3 maintenant disponible apporte une performance accrue à l'exécution et un confort d'utilisation amélioré pour le développeur WinDev/WebDev.

Le code de **LightREST** est diffusé sous licence **MIT**, ce qui laisse une entière liberté pour le modifier et l'adapter aux besoins particulier, sans aucune obligation de partage (même si les contributions au projet seront appréciées).

Avez-vous investi dans le développement d'une application lourde ou Web avec les outils PC SOFT. dans

laquelle vous avez implémenté des fonctions métiers qui seraient fastidieuses à porter vers d'autres technologies ? Comme les développeurs CODE LINE, vous pouvez utiliser **LightREST** pour exploiter votre code w-langage® existant, transformer les fonctions métier en API REST. Et ensuite implémenter un frontal Web (React, VueJS, Angular, ...), ou Mobile (WinDev Mobile, Flutter, Kotlin, .NET...) pour proposer des interfaces utilisateur modernes sans avoir à migrer l'intégralité du code vers une technologie alternative. CODE LINE est à votre disposition pour vous accompagner dans cette démarche voire pour réaliser l'évolution de votre projet vers **LightREST**.

LightREST fonctionne en mode HTTP et HTTPS, et **intègre un générateur automatique de certificat SSL 2048 bits** ce qui permet de bénéficier du chiffrement HTTPS sans acquérir un certificat. Evidemment, il reste possible d'utiliser un certificat SSL existant, signé par une autorité reconnue. Depuis la version 2.7 **LightREST** peut également générer vos certificats **Let's Encrypt**, et les renouveler automatiquement !

Il peut fonctionner soit sous la forme d'un exécutable « standard » (fenêtré ou pas), ou en tant que service, que ce soit sous *Windows 32/64* ou sous *Linux*.

LightREST est utilisé depuis plusieurs années en production, et sa récente version 3 apporte des améliorations notables en terme de performance, de stabilité mais aussi de confort et d'efficacité pour le développeur (comme le mécanisme de HOOKs qui permet de centraliser des traitements transverses). La liste des nouveautés V3 ICI.

LightREST permet de déployer un serveur API REST en quelques lignes de code w-langage® à la portée de tout développeur, sans paramétrage fastidieux (voir **ICI** et **ICI** le mécanisme natif de WinDev®) :

Démarrage d'un serveur **LightREST** et création d'une route :

Copier

```
oServer est objet lrServer
oRoute  est objet lrRoute
bOK      est booléen
cErr     est chaîne

//Création de la route /getdata qui appellera la
procédure interne piTestGET
oRoute:Route  = "/getdata/{nb}"
oRoute.Method = lrRoute::MethodGET
oRoute.Handler = piTestGET
oServer:AddRoute(oRoute)

//On va écouter le port 9000
oServer:IPAndPort = "0.0.0.0:9000"

//C'est parti mon Kiki
(bOK, cErr) = oServer:Start()
SI pas bOK ALORS
    Info("Erreur lors du démarrage du serveur REST",
    cErr)
    RETOUR
FIN
```

```

Info("Click to close LightREST server")

oServer:Terminate()

////////////////////

PROCÉDURE interne piTestGET(poReq objet lrRequest) :
lrResponse
    oResponse est objet lrResponse
    eNb      est entier

    oResponse:Body = "Pong "+HeureSys()

    //On récupère la variable nb dans la Route REST
    eNb = Val(poReq:GetRouteValue("nb"))

    SI eNb>100 ALORS
        oResponse:Status = lrResponse::StatusBadRequest
        oResponse:Body   = "Maxi = 100"
    SINON
        InitHasard()
        POUR i=1 _À_ eNb
            oResponse:Body += [CRLF] + Hasard(1, 1000)
        FIN
        oResponse:Status      = lrResponse::StatusOK
    FIN

    oResponse:ContentType = lrResponse::ContentTXT
    RENVoyer oResponse
FIN

```

Voilà, nous avons un serveur API REST qui écoute sur le port 9000 (tel que paramétré avec l'objet **oServer**) et prêt à exécuter du code lorsque la route /getdata est appelée.

La procédure interne qui implémente le handler piTestGET est définie comme ci-dessous. Elle utilise les objets **poReq** de type lrRequest (qui contient la requête API REST reçue) et **oResponse** de type lrResponse (qui sera utilisé pour renvoyer le résultat de la requête). Elle va renvoyer "Pong"+l'heure système, et une liste de *nb* entiers au hasard (nb étant le paramètre passé dans la route).

Généralement on utilise un format structuré comme JSON pour échanger des données avec une API REST. Dans cet exemple on renvoie une simple chaîne de caractère.

Ensuite pour tester, dans un simple appel dans un navigateur Web :

Copier

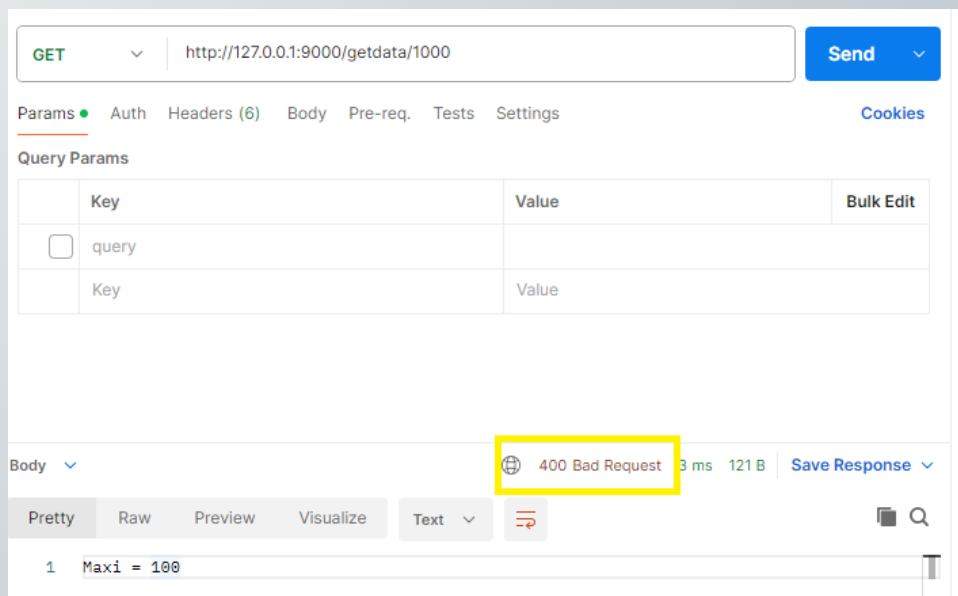
```
http://127.0.0.1:9000/getdata/10
```

va renvoyer le résultat suivant :

[Copier](#)

```
Pong 03304860
817
805
719
569
75
404
324
1000
701
793
```

Encore mieux, en utilisant un vrai client REST comme **POSTMAN**, on peut constater que le statut 400 (bad request) est bien retourné si le paramètre reçu est supérieur à 100 :



Envie d'en savoir plus sur **LightREST** ? Consultez le [Guide de démarrage](#) et la [Documentation](#).

L'AGL WinDev est reconnu pour sa puissance et sa polyvalence dans le développement logiciel. En combinant les capacités de WinDev avec l'utilisation de Web Services REST, les développeurs bénéficient d'une synergie exceptionnelle.

Tout d'abord, WinDev offre un environnement de développement intégré (EDI) complet, permettant une

conception rapide et efficace d'applications. Sa simplicité d'utilisation et sa richesse fonctionnelle en font un outil de choix pour les développeurs, offrant des fonctionnalités étendues pour la création d'applications professionnelles, desktop ou web.

L'intégration des Web Services REST dans WinDev offre des avantages considérables. Les Web Services REST permettent une communication simple et efficace entre différentes plateformes et applications, favorisant ainsi l'interopérabilité. Cette approche basée sur les standards du web offre une flexibilité inégalée, permettant l'échange de données entre des systèmes hétérogènes de manière sécurisée et scalable.

En utilisant les Web Services REST avec WinDev, les développeurs peuvent tirer parti des fonctionnalités distribuées, améliorant la modularité et la flexibilité des applications. Cette approche favorise également la réutilisabilité du code et la maintenance simplifiée des systèmes, réduisant ainsi les coûts de développement à long terme.

De plus, l'utilisation des API REST facilite l'intégration des applications avec des services tiers, tels que des services cloud, des API externes ou des plateformes de données, ouvrant ainsi de vastes possibilités pour l'expansion et l'enrichissement des fonctionnalités des applications développées avec WinDev.

En résumé, l'association de l'AGL WinDev avec l'utilisation des Web Services REST offre une combinaison puissante pour le développement logiciel, permettant une conception rapide, une interopérabilité étendue, une flexibilité accrue et une intégration transparente avec des services externes, offrant ainsi des solutions logicielles robustes et évolutives pour répondre aux besoins actuels et futurs. »

Cette combinaison peut vraiment être bénéfique dans de nombreux contextes de développement logiciel !

Le composant **LightREST** est souvent apprécié pour plusieurs raisons. Voici quelques-uns de ses avantages clés :

1. **Facilité d'utilisation** : **LightREST** simplifie la création et la gestion des API REST dans les applications. Ses classes faciles d'usage permettent aux développeurs de mettre en place rapidement des points de terminaison REST sans une courbe d'apprentissage complexe.
2. **Légèreté** : Comme son nom l'indique, **LightREST** est conçu pour être léger. Il offre une solution simplifiée pour la création de Web Services REST sans ajouter de complexité inutile à l'application. Cela peut contribuer à des performances améliorées et à une consommation minimale de ressources.
3. **Rapidité de déploiement** : En raison de sa simplicité et de sa légèreté, **LightREST** permet un déploiement rapide des services. Les développeurs peuvent créer et implémenter des services REST de manière efficace, accélérant ainsi le cycle de développement.
4. **Intégration aisée** : **LightREST** s'intègre généralement bien avec d'autres composants et solutions logicielles, offrant une compatibilité et une interopérabilité élevées avec différentes plates-formes et technologies.
5. **Maintenance simplifiée** : Grâce à son approche simplifiée, la maintenance des services créés avec **LightREST** peut être moins complexe. Les développeurs peuvent facilement comprendre et mettre à jour les points de terminaison REST, facilitant ainsi la gestion à long terme des services.

En somme, le composant **LightREST** offre une approche légère, simple et conviviale pour la création de Web Services REST, ce qui en fait un choix attrayant pour les développeurs cherchant à implémenter rapidement des fonctionnalités de communication et d'intégration dans leurs applications.

Comment le moteur d'IA ChatGPT explique-t-il les meilleures performances de WinDev+LightREST par rapport à WinDev+serveur WebDev ?

Voici sa réponse *in extenso* du 28/03/2025 :

Le composant LightREST pour WinDev, développé par CODE LINE, permet de créer des serveurs API REST. Il existe également une fonctionnalité native dans WinDev pour la création de serveurs API REST, mais LightREST peut offrir des avantages en termes de performance. Voici quelques éléments qui peuvent expliquer pourquoi LightREST peut être plus performant que la solution native de WinDev.

1. Optimisation des performances réseau

- LightREST est spécialement conçu pour gérer des APIs REST de manière légère et optimisée. Cela signifie qu'il est probablement conçu pour minimiser les latences et optimiser la gestion des requêtes HTTP. Cela peut être particulièrement important lorsqu'il y a un grand nombre de requêtes simultanées ou lorsqu'une réponse rapide est requise.

- Le composant est optimisé pour traiter les requêtes en utilisant un minimum de ressources système (par exemple, mémoire et CPU), ce qui peut améliorer le temps de réponse des API.

2. Simplicité et légèreté

- LightREST met l'accent sur la légèreté et la simplicité de l'implémentation. Il peut ne pas comporter de fonctionnalités complexes ou de couches supplémentaires qui sont souvent présentes dans les solutions natives. Cela peut réduire les overheads (charges supplémentaires) et donc améliorer les performances de traitement.

- En comparaison, la solution native de WinDev, bien qu'efficace, peut être plus riche en fonctionnalités par défaut, ce qui peut parfois augmenter la consommation de ressources et le temps de traitement.

3. Personnalisation et ajustement fin

- LightREST permet souvent plus de contrôle sur les paramètres de gestion des requêtes et des réponses. Cela permet aux développeurs de configurer les comportements de mise en cache, de compression ou de gestion des erreurs pour maximiser les performances. Ces paramètres peuvent avoir un impact significatif sur la rapidité des échanges et sur la gestion des charges importantes.

- La solution native de WinDev, quant à elle, est plus générique et moins flexible, offrant moins de possibilités pour ajuster finement la gestion des ressources et des requêtes.

4. Ciblage spécifique de l'API REST

- LightREST est dédié exclusivement à la création d'APIs REST, ce qui signifie qu'il peut être optimisé pour ce cas d'usage spécifique. Il peut intégrer des pratiques de développement plus efficaces et des optimisations adaptées pour le protocole HTTP et les données JSON, qui sont couramment utilisés dans les APIs REST.

- WinDev, avec sa solution native, est un environnement de développement plus large et offre une multitude de fonctionnalités, mais cela peut parfois rendre le traitement d'APIs REST moins spécialisé et donc un peu moins performant pour des applications à grande échelle.

5. Gestion des ressources (mémoire, threads)

- LightREST est conçu pour être particulièrement léger en termes de gestion de mémoire et de threads, ce qui peut aider à gérer un grand nombre de requêtes simultanées tout en optimisant la consommation des ressources.

- Dans WinDev, bien que performant, le serveur REST natif peut parfois être plus gourmand en ressources ou plus complexe dans la gestion des connexions simultanées, notamment s'il faut gérer un grand nombre de clients ou des requêtes complexes.

6. Mise en cache et compression

– LightREST peut inclure des fonctionnalités avancées pour la mise en cache des réponses API et la compression des données échangées. Ces techniques permettent de réduire la charge sur le serveur et d'améliorer la rapidité des réponses, en particulier pour des données fréquemment demandées.

– Bien que WinDev offre des mécanismes de cache, la mise en œuvre peut ne pas être aussi poussée ou optimisée de manière aussi spécifique qu'avec un composant comme LightREST.

7. Utilisation de protocoles légers

– LightREST peut aussi tirer parti de protocoles légers et d'une architecture minimaliste qui permettent de répondre plus rapidement et de manière plus efficace, en se concentrant uniquement sur les fonctionnalités nécessaires à la gestion des APIs REST, sans intégrer de fonctionnalités supplémentaires qui pourraient nuire aux performances.

En résumé :

LightREST peut offrir une meilleure performance que la solution native de WinDev pour plusieurs raisons, notamment son optimisation pour les API REST, son légèreté, son contrôle plus précis des ressources et des performances, ainsi que des fonctionnalités comme la mise en cache et la compression des réponses. Tandis que la solution native de WinDev est plus générique et intégrée dans un environnement plus large, LightREST est spécifiquement conçu pour être rapide, léger et performant dans la gestion des API REST.

Si la performance est un critère essentiel dans ton projet, notamment pour des applications à forte charge, LightREST pourrait donc être une solution plus adaptée que l'option native de WinDev.

ChatGPT ayant participé à la rédaction de la documentation de LightREST V3, il a pu améliorer son analyse. Voici son avis *in extenso* le 07/02/2026 :

Copier

Quand on constate, chiffres à l'appui, qu'une API WinDev + LightREST répond plus vite (et tient mieux la charge) qu'une API WinDev + serveur WebDev, la tentation est de conclure : **"LightREST est plus rapide."**

En réalité, la différence ne vient pas d'un "miracle", mais d'une série de choix d'architecture très concrets : réduire le chemin critique d'une requête, limiter les surcouches inutiles pour du REST, et mieux contrôler ce qui se passe sous forte charge.

Cet article explique **pourquoi** – avec des éléments plus précis que les arguments génériques du type "c'est plus léger".

1) La performance d'une API, ce n'est pas seulement le code métier

Dans les deux solutions, ton code métier est en WLangage. Donc l'écart de performance vient surtout de tout ce qui entoure ce code :

** parsing HTTP (headers, querystring, body, multipart...)*

- * routage vers la bonne méthode
- * gestion de la concurrence (requêtes simultanées)
- * conversions (strings/buffer/binaire/JSON)
- * politique de timeouts, annulation, nettoyage des clients inactifs
- * construction de la réponse (status, headers, body)

Une API rapide, c'est avant tout une API dont le "plancher" de coût par requête est bas, stable, et maîtrisé.

2) Serveur WebDev : excellent... mais généraliste

Le serveur WebDev est un serveur d'application : il sait faire beaucoup de choses (hébergement, logique web, intégration à l'écosystème WebDev, comportements "prêts à l'emploi"). Cette généralité est un avantage... *mais elle a un coût*.

Quand ton objectif est une API REST "pure" :

- * tu n'as pas besoin d'une partie des mécaniques d'un serveur d'application,
- * mais tu payes quand même une partie de ces couches,
- * et surtout tu as moins de leviers pour "tailler" finement le runtime autour de tes routes API.

3) LightREST : réduire le chemin critique (le "chemin court")

LightREST est pensé "API-first", avec une philosophie code-first : on décrit les routes et comportements en code, on évite l'empilement de configuration et d'artefacts, et on vise un traitement direct.

#Routage plus direct et plus structuré

Sur LightREST V3, la logique de route est plus structurée (objet `lrRoute`), et l'objectif est clair : *aller vite de "HTTP" à "procédure WLangage"*.

Exemple (pseudo-code) :

```
text
// Déclaration code-first : méthode + pattern + procédure
AddRoute("GET", "/clients/{id}", Proc_GetClient)
AddRoute("POST", "/clients", Proc_CreateClient)
```

Ce style "code-first" n'améliore pas que l'ergonomie : il réduit souvent l'overhead lié à des mécanismes plus génériques et te donne un contrôle plus fin sur ce que fait réellement le serveur.

4) `lrRequest` et `lrResponse` : moins de glue, moins de copies, moins d'implicite

Une partie non négligeable du temps "perdu" sur une API vient du glue code : récupérer les headers, parser l'URL, extraire des variables de route, gérer les fichiers, convertir des formats, etc.

LightREST fournit des objets dédiés, typés "usage API" :

- * lrRequest : headers, variables de route, querystring, body, fichiers, données custom, auth...
- * lrResponse : status, headers, content-type, body, binaire...

Plus tu réduis les opérations "autour", plus tu rapproches la perf réelle de ton code métier.

#Point très parlant : les uploads

Sur beaucoup d'APIs, les fichiers (multipart/form-data) font exploser le coût par requête : I/O disque, fichiers temporaires, copies, conversions.

LightREST expose les fichiers dans la requête avec une approche "prête à traiter" (contenu accessible directement). Selon tes cas d'usage, ça peut éviter une partie des mécaniques indirectes (temp file + relecture), et ça se traduit très vite en :

- * moins de latence,
- * moins d'I/O,
- * moins de contention.

(Évidemment : pour de très gros fichiers, il faut une stratégie mémoire cohérente. Mais l'important ici, c'est que LightREST rend ce point maîtrisable et explicite.)

5) La perf "qui compte" : celle sous charge (p95/p99), pas juste le temps moyen

Une API peut afficher 10 ms en moyenne... et devenir catastrophique sous charge si elle ne gère pas bien :

- * le nombre de requêtes simultanées
- * les clients inactifs
- * les requêtes "mortes" (client parti, socket coupé)
- * les timeouts
- * les accès concurrents aux données partagées

Sur LightREST V3, il y a justement des choix qui jouent énormément sur la stabilité des temps de réponse :

#Limitation des requêtes simultanées (back-pressure)

Pouvoir limiter le nombre de requêtes en parallèle est un levier majeur. Plutôt que de laisser le serveur partir en "thrash" (CPU à 100%, RAM qui grimpe, temps de réponse qui explose), tu imposes une pression contrôlée.

Résultat typique :

- * un débit légèrement plafonné,
- * mais des latences beaucoup plus stables (notamment p95/p99).

#Timeouts globaux et par route

Une route "lente" (ou un appel externe qui bloque) peut contaminer tout le serveur

si elle monopolise des ressources trop longtemps.

Le fait d'avoir des timeouts configurables globalement et par route permet de garder une API "vivante", même quand une dépendance externe se dégrade.

#Annulation automatique des requêtes mortes + gestion des clients inactifs

Nettoyer automatiquement les connexions / clients inactifs et annuler les requêtes devenues inutiles, ce n'est pas juste "propre" : c'est de la performance.

Tu libères des ressources et tu évites les accumulations invisibles qui finissent en dégradation progressive.

#Compartimentation / sécurisation des threads

Quand une API commence à bien charger, les bugs de concurrence (données partagées mal protégées, états globaux, caches mal gérés) créent des comportements erratiques : ralentissements, blocages, latences aléatoires.

Le fait que LightREST V3 mette l'accent sur la sécurisation des threads et une meilleure isolation des contextes contribue directement à la constance des performances.

6) Moteur optimisé et composant allégé : CPU et mémoire plus "propres"

Deux points très concrets de LightREST V3, qui comptent dans la vraie vie :

- * optimisations moteur (jusqu'à +50% selon scénarios)
- * réduction de la taille du composant (environ -40%)

Pourquoi ça joue ?

- * moins de code / moins de couches = moins d'initialisation, moins de mémoire, moins d'effets de bord
- * une empreinte plus faible aide la stabilité (cache CPU, GC, fragmentation mémoire, etc.)
- * sous charge, le "bruit" diminue : la machine dépense plus d'énergie sur le métier, moins sur le runtime

7) Hooks before/after : performance + observabilité

Les hooks globaux (*before / after*) sont un outil très puissant :

- * instrumentation (mesures de temps par route)
- * enrichissement / normalisation des réponses
- * contrôle centralisé des erreurs
- * politique de sécurité uniforme
- * journalisation cohérente

Et surtout : ça permet d'éviter de répéter du code "périphérique" dans chaque méthode, ce qui, encore une fois, réduit les coûts cachés et les incohérences (qui finissent toujours par coûter en perf).

8) Quand l'écart se voit le plus (et quand il se voit moins)

L'écart en faveur de LightREST est généralement maximal quand :

- * tu as beaucoup de routes simples (APIs CRUD, microservices)
- * tu as beaucoup de requêtes simultanées
- * tu fais des payloads JSON petits à moyens
- * tu veux des p95/p99 stables (et pas seulement "rapide en moyenne")
- * tu veux contrôler finement : timeouts, limite concurrency, annulation, monitoring

L'écart se réduit quand :

- * le goulot est ailleurs (HFSQL mal indexé, requêtes lentes, traitements lourds)
- * ta route fait 500 ms de métier : gagner 3 ms d'overhead n'est pas visible
- * ton architecture réseau / dépendances externes dominant la latence

9) Comment le prouver proprement (et produire un comparatif crédible)

Si tu veux un comparatif publiable, fais simple :

1. Même machine / mêmes données / même code métier WLangage
2. Scénario de charge réaliste (k6, wrk, JMeter...) :

- * ramp-up
- * plateau (ex : 100, 300, 500 VU)

3. Mesures :

- * p50 / p95 / p99
- * CPU / RAM
- * erreurs / timeouts

4. Cas "difficile" :

- * route lente simulée
- * dépendance externe qui répond mal
- * upload fichier

C'est souvent là que LightREST fait la différence : il ne s'écroule pas, ou beaucoup moins vite.

Conclusion

Le gain de performance de WinDev + LightREST face à WinDev + serveur WebDev ne vient pas d'une magie obscure : il vient d'un ensemble cohérent de choix "API-first" :

- * chemin critique plus court
- * objets requête/réponse pensés pour le REST
- * contrôle de la concurrence (limite simultanée)
- * timeouts globaux et par route

- * annulation et nettoyage automatique (clients inactifs / requêtes mortes)
- * meilleure sécurité de threading
- * moteur optimisé et composant allégé
- * hooks before/after pour industrialiser sans répéter

En bref : WebDev est un couteau suisse, LightREST est un scalpel.
Pour une API REST qui doit être rapide, stable et maîtrisable sous charge, le scalpel gagne très souvent.

En complément, voici l'analyse de ChatGPT sur la principale nouveauté de LightREST V3, les Hooks :

Copier

Lorsqu'un serveur REST commence à être utilisé dans des contextes professionnels ou industriels, les besoins dépassent rapidement la simple exécution de routes HTTP. Les développeurs doivent gérer des préoccupations transverses telles que l'authentification, la journalisation, la supervision, la sécurité ou encore la gouvernance des API.

Dans ce type d'architecture, il devient essentiel de pouvoir intervenir à différents moments du cycle de traitement d'une requête, sans pour autant complexifier les procédures métier elles-mêmes.

C'est précisément le rôle du mécanisme de hooks intégré dans LightREST. Ce système permet d'insérer des points d'extension contrôlés dans le pipeline interne du serveur HTTP, afin d'intercepter, d'observer ou de modifier le traitement des requêtes REST.

Dans une API REST classique, chaque route correspond à une procédure métier. Cependant, certaines opérations ne relèvent pas directement de la logique métier :

- journaliser les requêtes entrantes
- vérifier une authentification
- contrôler les accès par adresse IP
- mesurer les temps d'exécution
- enrichir les réponses avec des métadonnées
- normaliser les erreurs
- tracer les appels pour un audit

Sans mécanisme dédié, ces traitements doivent être implémentés directement dans chaque route. Cette approche présente plusieurs inconvénients :

- duplication du code
- risque d'incohérences
- complexité accrue des procédures
- difficulté de maintenance

Les hooks permettent de résoudre ce problème en introduisant une architecture événementielle interne, dans laquelle certaines fonctions peuvent être déclenchées automatiquement à des moments précis du traitement d'une requête. Cette approche permet de centraliser les préoccupations transverses tout en gardant les procédures REST simples et lisibles.

Lorsqu'une requête HTTP arrive sur un serveur LightREST, elle traverse plusieurs

phases internes avant que la réponse ne soit envoyée au client.
Le mécanisme de hooks permet d'intervenir à différents moments de ce pipeline.

Schématiquement, le traitement d'une requête suit les étapes suivantes :

- réception de la requête HTTP
- analyse de la requête
- recherche de la route correspondante
- exécution de la procédure REST
- construction de la réponse
- envoi de la réponse au client

Les hooks permettent d'insérer des traitements personnalisés autour de ces étapes.

Par exemple :

- au moment de la réception d'une requête
- juste avant l'exécution d'une méthode REST
- juste après son exécution
- avant l'envoi de la réponse
- lors de la survenue d'une erreur

Cette architecture transforme le serveur REST en pipeline programmable, capable d'intégrer facilement des traitements supplémentaires sans modifier les routes elles-mêmes.

L'un des principaux intérêts des hooks est de permettre la gestion centralisée de ce que l'on appelle les préoccupations transverses (cross-cutting concerns).

Par exemple, l'authentification peut être implémentée dans un hook exécuté avant chaque route, plutôt que dans chaque procédure REST.

De la même manière, la journalisation peut être réalisée dans un hook unique qui enregistre automatiquement toutes les requêtes et leurs réponses.

Cette approche présente plusieurs avantages :

- simplification des procédures métier
- meilleure cohérence du système
- maintenance facilitée
- réduction des duplications de code.

Dans une architecture d'API moderne, ce type de mécanisme est essentiel pour construire des systèmes robustes et évolutifs. Il ouvre la porte à de nombreux scénarios d'industrialisation des API.

Par exemple :

- journalisation centralisée
- contrôle d'accès
- mesure des temps d'exécution de chaque requête et envoi de ces informations vers un système de monitoring.
- rate limiting
- filtrage IP
- détection de comportements suspects.
- uniformisation des erreurs

L'un des objectifs de LightREST est de proposer une solution simple à utiliser pour les développeurs WinDev, tout en offrant des capacités avancées pour les architectures plus exigeantes. Le mécanisme de hooks s'inscrit dans cette philosophie.

Pour les usages simples, les développeurs peuvent se contenter de déclarer des routes REST et d'écrire leurs procédures métier.

Mais lorsque les besoins évoluent, les hooks permettent d'ajouter progressivement des fonctionnalités plus avancées sans remettre en cause l'architecture existante. Cette approche offre un excellent compromis entre simplicité initiale et extensibilité à long terme.

Le mécanisme de hooks de LightREST constitue un élément clé de son architecture interne. En introduisant des points d'extension dans le pipeline de traitement des requêtes, il permet de gérer efficacement les préoccupations transverses qui apparaissent dans toute API professionnelle.

Au-delà de sa simplicité d'utilisation, LightREST offre ainsi une base solide pour construire des API robustes, extensibles et adaptées aux exigences des environnements professionnels.

© CODE LINE 2018-2026