Engineer

Couche serveur

la partie d'un système que le lecteur ne charge jamais

Ce que c’est

La couche serveur est la partie d’un système logiciel exécutée sur des machines que son exploitant contrôle : elle stocke les données, applique les règles sur les actions permises et répond aux requêtes des navigateurs, applications et autres services. Son nom anglais backend s’écrit aussi back end ou back-end, et le même territoire est souvent appelé côté serveur.

En HTTP, elle correspond à ce que la RFC 9110 nomme serveur d’origine, « un programme capable de produire des réponses faisant autorité pour une ressource cible donnée ». L’Architecture of the World Wide Web du W3C ajoute le point de vue du lecteur : une URI identifie une ressource, et ce qui arrive au navigateur en est une représentation, des données encodant son état. Un client voit une page de commande ; la couche serveur détient la commande, le stock et le paiement, et décide de la réponse.

Cette couche expose généralement une API, ensemble de points d’entrée HTTP qu’une interface, une application mobile ou le système d’une autre entreprise peut appeler. L’API est un contrat : chemins existants, entrées acceptées, résultats et erreurs possibles. OpenAPI Specification est le moyen courant d’écrire ce contrat comme description indépendante du langage, à partir de laquelle des outils génèrent clients, documentation et tests. Derrière l’API, le travail est généralement séparé en traitement des requêtes, logique métier et stockage.

Utilisez le concept pour prendre une décision

Utilisez « Qu'est-ce que la couche serveur ? » pour éclairer une décision réelle. Nous pouvons vous aider à relier ce concept au système que vous construisez ou achetez. La partie invisible pour les utilisateurs : où résident les données, quelles règles s'appliquent et comment les autres systèmes peuvent y accéder.

Découvrir nos prestations Contactez-nous →

Vérifiez les hypothèses derrière le terme

Appuyez-vous sur l'explication et les réalisations liées pour formuler les questions de votre projet. Nous pouvons examiner les hypothèses avec vous avant le choix d'un outil. 6 pipelines ETL/ELT conçus et exploités — 6 pipelines ETL sur trois moteurs de bases de données

6 pipelines ETL/ELT conçus et exploités Contactez-nous →

Pourquoi c’est important

La première question est de savoir si le système nécessite réellement une couche serveur active à chaque requête. Une page identique pour tous les visiteurs peut être générée une fois et servie comme fichier. Une couche active se justifie lorsque la requête doit modifier quelque chose ou dépend de son auteur : commande, connexion, recherche dans des données privées. Elle porte aussi un coût permanent, car un logiciel répondant à Internet doit être sécurisé, mis à jour et payé.

La deuxième question est la confiance. Les machines contrôlées par l’exploitant sont le seul endroit où imposer une règle. Toute vérification de l’interface peut être évitée en envoyant directement des requêtes. Chaque règle importante, du prix aux permissions, est donc vérifiée côté serveur pour l’utilisateur connecté et l’enregistrement précis. OWASP Application Security Verification Standard en fait des exigences sur lesquelles construire et tester.

La troisième question est l’emplacement de la logique. Une règle dispersée entre gestionnaire, classe de service et déclencheur de base comporte trois lieux à modifier et trois copies susceptibles de diverger. Certaines équipes gardent les règles dans le code applicatif et utilisent la base pour stocker ; d’autres y placent règles et façonnage des données, en gardant une API légère. Les deux approches fonctionnent si chaque règle a un seul domicile.

La quatrième question est le contrat client. HTTP donne à chaque réponse un code de statut dont le sens est défini, utilisé par clients, mandataires et supervision. 422 Unprocessable Content indique une requête syntaxiquement correcte dans un format pris en charge, mais sémantiquement invalide, comme une réservation dans le passé. La RFC 9457 ajoute un corps standard d’erreur, les détails de problème : un petit objet JSON contenant type, titre, statut, détail et instance, envoyé comme `application/problem+json` . Un client peut ainsi distinguer les échecs sans analyser la prose.

À quoi cela ressemble en pratique

Le trajet habituel de la base au navigateur est un relais de transformations : l'API convertit les lignes en structures, les remodèle, puis les sérialise en JSON. Sur une plateforme, la couche serveur laisse la base construire chaque réponse et la transmet intacte. Les fonctions PostgreSQL assemblent la réponse complète en JSON là où résident déjà les données. Le gestionnaire Go la reçoit en `json.RawMessage` et la transmet sans modification. Une petite fonction utilitaire `QueryJSON` en a fait la voie la plus simple. DTO, convertisseurs et structures de réponse ont disparu, les gestionnaires ont considérablement raccourci et tout changement de résultat d'une fonction atteint directement le client sans édition des gestionnaires.

Les requêtes entrantes reçoivent le même traitement à source unique, grâce à un contrat partant de la base vers l'extérieur. Côté serveur, le schéma et ses fonctions définissent les formes, et les types Go définissent l'API. Huma produit la description OpenAPI à partir de ces types, puis les 44 076 lignes de TypeScript typé du client de l'interface sont générées à partir de cette description. Les limites des formulaires viennent du même endroit plutôt que d'être ressaisies dans un validateur. Chaque étape possède un contrôle : un champ ne peut pas changer d'un seul côté, car l'écart bloque la compilation quelques minutes après. La contrepartie est la régénération : tout changement d'une forme définie côté serveur doit descendre la chaîne en régénérant chacun des maillons suivants.

Au stade de conception consigné, ce site a fait le choix opposé. Il était entièrement généré à la compilation et ne nécessitait aucun backend applicatif au moment des requêtes. Ses quatre exigences, dont le fonctionnement en trois langues et l'absence de service applicatif facturé séparément, ont conduit à ce choix : un backend applicatif à l'exécution aurait dû être sécurisé, mis à jour et expliqué sur une page de confidentialité. En dehors de l'exception JavaScript limitée et consignée pour un filtre de portfolio, les fonctions interactives devaient être résolues à la compilation ou omises. Cela décrit le stade de conception consigné, et non l'inventaire ou l'implémentation actuels du site ; cela ne signifie pas que l'hébergement était gratuit.

Où les problèmes surviennent

Les défauts restent souvent silencieux jusqu'à l'arrivée d'une requête que personne n'avait prévue.

Un utilisateur modifie un enregistrement appartenant à quelqu'un d'autre. Le point d'entrée vérifie sa connexion, puis charge l'identifiant de l'URL : changer `/invoices/1041` en `/invoices/1042` suffit. Vérifiez côté serveur propriété ou permission de chaque objet touché, en un endroit commun à tous les gestionnaires. Gardez un test qui demande l'enregistrement d'un autre compte et attend 403 ou 404.

Une action se produit deux fois. Un lien d'annulation par GET est suivi par un robot ou un navigateur préchargeant les pages, et l'action s'exécute sans clic. La RFC 9110 définit GET, HEAD, OPTIONS et TRACE comme sûrs, pour permettre exploration et préchargement sans risque de dommages. Autre voie : un paiement dépasse le délai, le client le renvoie, et la personne paie deux fois. Placez tout changement d'état derrière POST, PUT, PATCH ou DELETE et donnez aux écritures susceptibles d'être retentées une clé mémorisée côté serveur, pour que la seconde copie renvoie le premier résultat.

Toutes les erreurs reviennent en 500 ou en 200 avec un message dans le corps. Un gestionnaire général traite tous les échecs pareil : la supervision compte les fautes de saisie comme pannes, et les reprises répètent des requêtes qui échoueront toujours. Associez chaque échec au statut qui le décrit : 400 ou 422 pour une entrée invalide, 401 et 403 pour identité et permissions, 404 pour ressource absente, 409 pour conflit, et 5xx seulement pour les défauts du serveur. Envoyez les détails au format standard :

```
{
    "type": "https://example.com/problems/booking-in-past",
    "title": "Booking date is in the past",
    "status": 422,
    "detail": "The date 2026-10-01 is before today.",
    "instance": "/bookings/8812"
}
```

La RFC 9457 précise que ces détails « ne constituent pas un outil de débogage de l'implémentation sous-jacente » : traces de pile et SQL restent donc dans les journaux serveur.

Les pages ralentissent avec les données. Un gestionnaire charge les commandes puis effectue ensuite une requête pour récupérer le client de chacune : 50 commandes produisent 51 allers-retours à la base. Cela paraît rapide avec dix lignes en développement et se traîne avec dix mille en

production. Récupérez les données liées par une jointure ou laissez la base assembler toute la réponse. Surveillez le nombre de requêtes par demande aussi attentivement que la durée de réponse.

Pour aller plus loin

- Architecture of the World Wide Web, Volume One — recommandation W3C sur ressources, représentations et interactions sûres ou non.
- RFC 9110 : HTTP Semantics — norme IETF des méthodes, statuts et propriétés de sûreté ou d'idempotence.
- OpenAPI Specification — format indépendant du langage décrivant une API HTTP, lu par les personnes et générateurs.
- RFC 9457 : Problem Details for HTTP APIs — format JSON standard des erreurs lisibles par machine.
- OWASP Application Security Verification Standard — exigences de sécurité pour construire et tester des applications Web.

Glossaire

Choisissez selon vos contraintes réelles

Présentez le processus de travail, les interfaces existantes et les utilisateurs qui ont besoin du changement. Nous pouvons définir une livraison testable et une transmission permettant à votre équipe d'assurer la maintenance des éléments livrés.

Consulter les tarifs Contactez-nous →

Rendez la terminologie concrète

Demandez comment « Qu'est-ce que la couche serveur ? » s'applique à votre organisation. Un exemple concret tiré de votre système actuel suffit pour engager la discussion.

Découvrir nos prestations

welcome@engineer.company +45 50 10 61 84 Copenhague, Danemark

Emportez cette page avec vous

Fiche de contact

Créé avec soin et amour par D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Droit d'auteur © 2025 – aujourd'hui · Engineer ApS · Tous droits réservés.

Imprimé depuis https://engineer.company/fr/glossary/backend/