Engineer

Бекенд

та частина системи, яку читач ніколи не завантажує

Що це таке

Бекенд — це частина програмної системи, яка працює на машинах під контролем її оператора: він зберігає дані, застосовує правила про те, кому що дозволено, і відповідає на запити від браузерів, застосунків та інших сервісів. Англійською його також пишуть back end або back-end, а ту саму територію часто називають серверною стороною, server side.

У термінах HTTP бекенд — це те, що RFC 9110 називає origin server: програма, яка може створювати авторитетні відповіді для певного цільового ресурсу. Документ W3C Architecture of the World Wide Web додає погляд читача. URI ідентифікує ресурс, а до браузера вирушає його представлення, дані, що кодують інформацію про стан ресурсу. Клієнт бачить сторінку замовлення, а бекенд зберігає замовлення, залишки на складі й оплату та вирішує, що потрапить у відповідь.

Зазвичай бекенд надає API, набір HTTP-ендпоінтів, які може викликати фронтенд, мобільний застосунок або система іншої компанії. API — це контракт: які шляхи існують, що приймає кожен із них, що повертає і які помилки може видати. OpenAPI Specification — поширений спосіб записати цей контракт у вигляді опису, незалежного від мови програмування, з якого інструменти генерують клієнти, документацію й тести. За API робота зазвичай поділена на шари: обробку запитів, бізнес-логіку й зберігання.

Застосуйте поняття до рішення

Застосуйте «Що таке бекенд?» до конкретного рішення. Ми можемо пов'язати поняття із системою, яку ви створюєте або купуєте. Та частина, якої користувач не бачить: де зберігаються дані, які діють правила і як інші системи можуть із нею взаємодіяти.

Переглянути наші послуги Зв'язатися з нами →

Перевірте припущення за терміном

Використайте пояснення та пов'язані роботи, щоб сформулювати запитання для проєкту. Ми можемо перевірити припущення разом до вибору інструмента. Спроєктувала й підтримувала 6 пайплайнів ETL/ELT — 6 пайплайнів ETL на трьох рушіях баз даних

Спроєктувала й підтримувала 6 пайплайнів ETL/ELT Зв'язатися з нами →

Чому це важливо

Перше питання — чи потрібен системі взагалі бекенд, що працює в момент запиту. Сторінку, однакову для кожного відвідувача, можна згенерувати один раз і віддавати як файли. Бекенд, що працює постійно, виправдовує себе, коли запит має щось змінити або залежить від того, хто його надіслав: замовлення, вхід в обліковий запис, пошук у приватних даних. До того ж він коштує грошей щодня, бо програму, яка відповідає на запити з інтернету, треба захищати, оновлювати й оплачувати.

Друге — довіра. Бекенд працює на машинах, які контролює оператор, тому саме там правило можна справді забезпечити. Будь-яку перевірку у фронтенді обійде той, хто надсилає запити напряму, тож кожне важливе правило, від ціни до дозволу, перевіряють на сервері щодо користувача, який увійшов, і конкретного запису. OWASP Application Security Verification Standard перетворює це на вимоги, за якими команда може будувати й тестувати.

Третє — де живе логіка. Правило, розкидане між обробником, сервісним класом і тригером у базі даних, має три місця для змін і три копії, які розходяться. Одні команди тримають правила в коді застосунку, а базу використовують для зберігання; інші переносять правила й формування даних у базу і тримають API тонким. Працювати може і те, і те, якщо кожне правило має один дім.

Четверте — контракт із клієнтами. HTTP дає кожній відповіді код стану з визначеним значенням, і клієнти, проксі та інструменти моніторингу діють відповідно до нього. Код 422 Unprocessable Content означає, що запит синтаксично правильний і в підтримуваному форматі, але семантично хибний, як-от бронювання з датою в минулому. RFC 9457 додає стандартний формат для помилок, problem details: невеликий об’єкт JSON із полями type, title, status, detail та instance, що надсилається як `application/problem+json`, щоб клієнт розрізняв помилки без розбору тексту.

Як це виглядає на практиці

Бекенд однієї платформи доручає базі даних будувати кожну відповідь і передає її без змін. Звичний шлях від бази до браузера сторінка називає естафетою перетворень: API розбирає рядки у структури, перекроює їх і знову серіалізує в JSON. Тут функції PostgreSQL збирають усю відповідь як JSON там, де вже лежать дані, а обробник на Go отримує її як `json.RawMessage` і пересилає далі як є. Невелика допоміжна функція `QueryJSON` зробила цей шлях найпростішим. DTO, мапери й структури відповідей зникли, код обробників став значно коротшим, а зміна в тому, що повертає функція, доходить до клієнта напряму, і правити код обробника не доводиться.

Для вхідних запитів та сама ідея одного джерела означає, що контракт починається з бази даних і поширюється назовні. Схема та її функції задають структури даних, типи Go задають API, Huma генерує з них опис OpenAPI, а з цього опису генерується типізований клієнт TypeScript на 44 076 рядків. Обмеження, які інтерфейс застосовує до форми, беруться з того самого джерела, і вручну їх у валідатор ніхто не переносить. Кожен перехід охороняється, тож поле не може змінитися лише з одного боку: розбіжність ламає збирання за кілька хвилин після зміни. Ціна, яку називає сторінка, — крок генерації в самому центрі процесу.

На задокументованому етапі проєктування цей сайт обрав протилежний підхід. Він повністю генерувався під час збирання й не потребував бекенду застосунку для обробки запитів. Чотири вимоги, зокрема робота трьома мовами й відсутність окремо оплачуваного сервісу застосунку, привели до цього вибору: такий бекенд довелося б захищати, оновлювати й описувати на сторінці про конфіденційність. Окрім зафіксованого обмеженого винятку JavaScript для фільтра портфоліо, інтерактивні функції потрібно було реалізовувати під час збирання або не додавати. Це опис задокументованого етапу проєктування, а не нинішнього складу чи реалізації сайту; він не означає, що хостинг був безкоштовним.

Де виникають проблеми

Збої бекенду зазвичай мовчать, доки не надійде запит, якого ніхто не передбачив.

Користувач змінює запис, що належить комусь іншому. Ендпоінт перевіряє, що той, хто викликає, увійшов, а потім завантажує будь-який ID, названий в URL, тож достатньо змінити в запиті `/invoices/1041` на `/invoices/1042`. Власність або дозвіл треба перевіряти на сервері для кожного об'єкта, якого торкається запит, в одному місці, крізь яке проходить кожен обробник, і мати тест, що просить запис іншого облікового запису й очікує 403 або 404.

Дія виконується двічі. Посилання, що скасовує щось через GET-запит, відкриває краулер або браузер, який завантажує сторінки наперед, і дія відбувається без жодного кліку. RFC 9110 визначає GET, HEAD, OPTIONS і TRACE як безпечні (safe) методи саме для того, щоб пошукові роботи й попереднє завантаження могли працювати без ризику завдати шкоди. Інший шлях — повторна спроба: платіжний запит перевищує час очікування, клієнт надсилає його знову, і покупець платить двічі. Кожну зміну стану варто ставити за POST, PUT, PATCH або DELETE, а записам, які можуть повторитися, давати ключ, який сервер пам'ятає, щоб друга копія повертала результат першої.

Кожна помилка повертається як 500 або як 200 із повідомленням про помилку всередині. Один універсальний обробник повідомляє про всі збої однаково, тож моніторинг рахує друкарські помилки як аварії, а логіка повторних спроб повторює запити, які щоразу завершаться помилкою. Кожен збій варто зіставити з кодом стану, що його описує: 400 або 422 для поганого введення, 401 і 403 для ідентичності й дозволів, 404 для відсутнього ресурсу, 409 для конфлікту і 5xx лише для власних збоїв сервера. Подробиці надсилають як problem details:

```
{
    "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"
}
```

RFC 9457 зазначає, що problem details не є інструментом налагодження базової реалізації, тому стектрейси й SQL залишаються в журналах сервера.

Сторінки сповільнюються з ростом даних. Обробник завантажує список замовлень, а потім для кожного окремим запитом отримує дані клієнта, тож сторінка з 50 замовлень робить 51 звернення до бази даних. У розробці з десятьма рядками це здається швидким, а в продакшні з десятьма тисячами гальмує. Пов'язані дані варто отримувати одним запитом із join або доручити базі зібрати всю відповідь, і стежити за кількістю звернень до бази на один запит так само уважно, як за часом відповіді.

Що почитати далі

- Architecture of the World Wide Web, Volume One — рекомендація W3C про ресурси, представлення та безпечні й небезпечні взаємодії.
- RFC 9110: HTTP Semantics — стандарт IETF про методи, коди стану і те, що робить запит безпечним або ідемпотентним.
- OpenAPI Specification — незалежний від мови формат опису HTTP API, який читають і люди, і генератори коду.
- RFC 9457: Problem Details for HTTP APIs — стандартна форма JSON для машинозчитуваних відповідей про помилки.
- OWASP Application Security Verification Standard — перелік вимог безпеки для створення й тестування вебзастосунків.

Словник

Обирайте за фактичними умовами

Опишіть робочий процес, наявні інтерфейси та користувачів, яким потрібна зміна. Ми можемо визначити перевірюваний результат і спосіб передачі, який дасть вашій команді змогу підтримувати його.

Дивитися, скільки це коштує Зв'язатися з нами →

Зробіть термінологію практичною

Запитайте, як «Що таке бекенд?» стосується вашої організації. Конкретного прикладу з поточної системи достатньо для початку розмови.

Переглянути наші послуги

welcome@engineer.company +45 50 10 61 84 Копенгаген, Данія

Візьміть цю сторінку із собою

Візитівка

Дбайливо зроблено з любов'ю — D.

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

Авторське право © 2025 – дотепер · Engineer ApS · Усі права захищено.

Надруковано з https://engineer.company/uk/glossary/backend/